RightWebMap
User Reference Guide

5 April 2005
Release 7.0

ER Mapper
Helping people manage the earth

Asia/Pacific :

ER Mapper
Level 2
87 Colin Street
West Perth
Western Australia 6005
Telephone:+61 8 9388-2900
Facsimile: +61 8 9388-2901

Europe, Africa and Middle East:
ER Mapper
Blenheim House
Crabtree Office Village
Eversley Way, Egham
Surrey, TW20 8RY, UK
Telephone: +44 1784 430-691
Facsimile: +44 1784 430-692

Americas:
ER Mapperg
13400 Sabre Springs
Parkway
Suite 150, San Diego
CA 92128, USA
Telephone: +1 858 391-5638
Facsimile: +1 858 391-5649

Copyright information

Revision History

Revision	Date	Comments
Release 7.0	5 April 2005	JPEG 2000 support added
Release 2.2	April 2004	Added EVIEW 4 EXL Access Method
Release 2.1		
Release 2.0		
Release 1.9		
Release 1.5		

Table of Contents

1

Introduction

RightWebMap is a client-side solution that integrates multiple GIS and image services into a single application view. It integrates different data sources, including ArcIMS, OGC WMS, ECWP and other map servers.

This product provides an efficient, quick-deployment, enterprise-scale integration environment for your organisation.

This introduction provides a broad overview of the following:

- Whats New
- Upgrading
- System Requirements
- Licensing

Use RightWebMap to advantage in your organisation by making use of the following features:

- Real time roam and zoom - no more "click - wait - click - wait" user interface.
- Integration of multiple web map servers into a single view for users.
- Integrates OGC/WMS, ECWP (Image Web Server), ArcIMS (ArcXML), Mapserver, MapGuide, MapXtreme and MapPoint .NET servers.
- Easy to add support for other types of servers
- Fast raster imagery layers using ECWP streaming imagery protocol.
- Easy to use and intuitive user interface
- New web maps can be created with only a few lines of JavaScript
- Open client interface
- Hides differences between browsers and user platforms from developers.

- Reduces server loads
- Interactive layering, stacking and transparency between map layers from different servers.

Whats new

Version 7.0

New ECW JPEG 2000 support

RightWebMap can access industry standard ECW files and ISO standard JPEG 2000 files when served from an Image Web Server. You can choose between storing imagery in the industry standard ECW format, or the ISO standard JPEG 2000 format.

Our ECW JPEG 2000 technology is specifically designed to cater for the needs of large geospatial deployments. Single images can be 1,000's of gigabytes in size.

New Access lossless imagery

Support for compressed lossless and lossy image datasets. You can now use RightWebMap for applications that require pixel-for-pixel fidelity such as feature classification or military implementations.

New Streaming JPEG 2000 via ECWP

RightWebMap can access JPEG 2000 files faster than you thought possible using Image Web Server's streaming ECWP protocol. ECWP is a proven technology that can efficiently stream even the largest imagery datasets.

Enhanced RightWebMap Engine

You can expect even better performance with refinements to the underlying RightWebMap engine.

New Added Datums and Projections

New datums and projections have been added to this release including the Canadian NTv2, giving you greater flexibility in the image datasets you deploy.

Additional enhancements

- Improved plug-in capabilities.
- Java plug-in enhanced.

Version 2.2

- Various bug fixes

- Added EVIEW 4 EXL Access Method

Version 2.1

- rApp PopupWindow now uses a more generic test to see if the error message is a HTML formatted string.
- Xor layers now must cover at least 3 sides of the map area before being considered.
- Rebuilds all control layers when map projection changes

Version 2.0

- Added new rX[] associative array to uniquely identify objects for DHTML creation. This enables objects to be created at any time and also deleted without impacting on other objects.
- Removed the definition of the *this.myself* value from the init() method into the object creation function, using the new *rX[]* array identifying all created objects.
- Removed *doneInit* from rService object as no longer required
- Moved create cacheid for eview access interface into the rEview3Access object.
- Moved status line creation from init() function to build() function
- Converting multi-choice lists in Eview protocol to string field if more than 100 entries as multi-view lists become unusable for users.
- rSearch Number type fields now leave field default as blank instead of "0" when default is not specified to make fields look cleaner.
- Expanded layer meta data response to provide more information and cleaner layout.
- Added new global: *rAccess[]* which is an associated array of RAccess objects. Each access protocol adds it's creation function to the rAccess array. New access methods can now be added via external .js include files (which should be included after rightwebmap.js).
  Important: access objects prior to version 2.0 need updating to add themselves to the *rAccess* object (all access objects included in the rightwebmap.js file for 2.0 have been updated).
- Enhanced RealtimeGPS access interface and added example database for GPS tracking

Version 1.9

- Improved ArcXML interface:
- Reversed order of layers display
- Display layers only when in scale range
- Layers tab only displays layers when layer.hidden == false
- Cleaned up demo RightWebMap.htm demo page
- Redirection for plugin download now cleanly returns to RightWebMap page

Version 1.5

- Query rectangle, polygon and circle as well as query point

- Tool bars can be any height for frame style layout (not limited to one line)
- Tool bars have a new addSpace("Row" | "Column") method to control spacing and layout of items
- Smarter JPEG transparency using new 1.8 control capabilities
- Refined WMS protocol interface (and requires 1.1 or higher WMS servers)
- Refined and improved styles

Upgrading

If you are already using Version 2.x of RightWebMap, you need do nothing extra to upgrade. Simply install RightWebMap normally. Libraries from previous versions of RightWebMap have the same names as before. Any file path settings should remain the same. Otherwise, simply set your file paths to the location of the new RightWebMap and rebuild your application.

System requirements

RightWebMap ships with pre-built binaries that require one of the following operating systems:

- Windows 95, 98SE / ME, NT4, 2000, XP and 2003 Server.

All available operating system updates should be applied to ensure correct operation of RightWebMap.

Note: Windows 95 is no longer supported.

Licensing

If you own a license of Image Web Server, you will find an evaluation copy of RightWebMap installed with it. This version may not be used for commercial purposes.

You can also access this version by using the following URL:

http://<your server>/rightwebmap/rightwebmap.htm

Replace <your server> with the name of the server on which you have Image Web Server installed.

To use RightWebMap in a commercial application, visit http://www.ermapper.com, or contact an ER Mapper office closest to you for a standalone evaluation copy of RightWebMap or for help with integration issues.

2

Overview

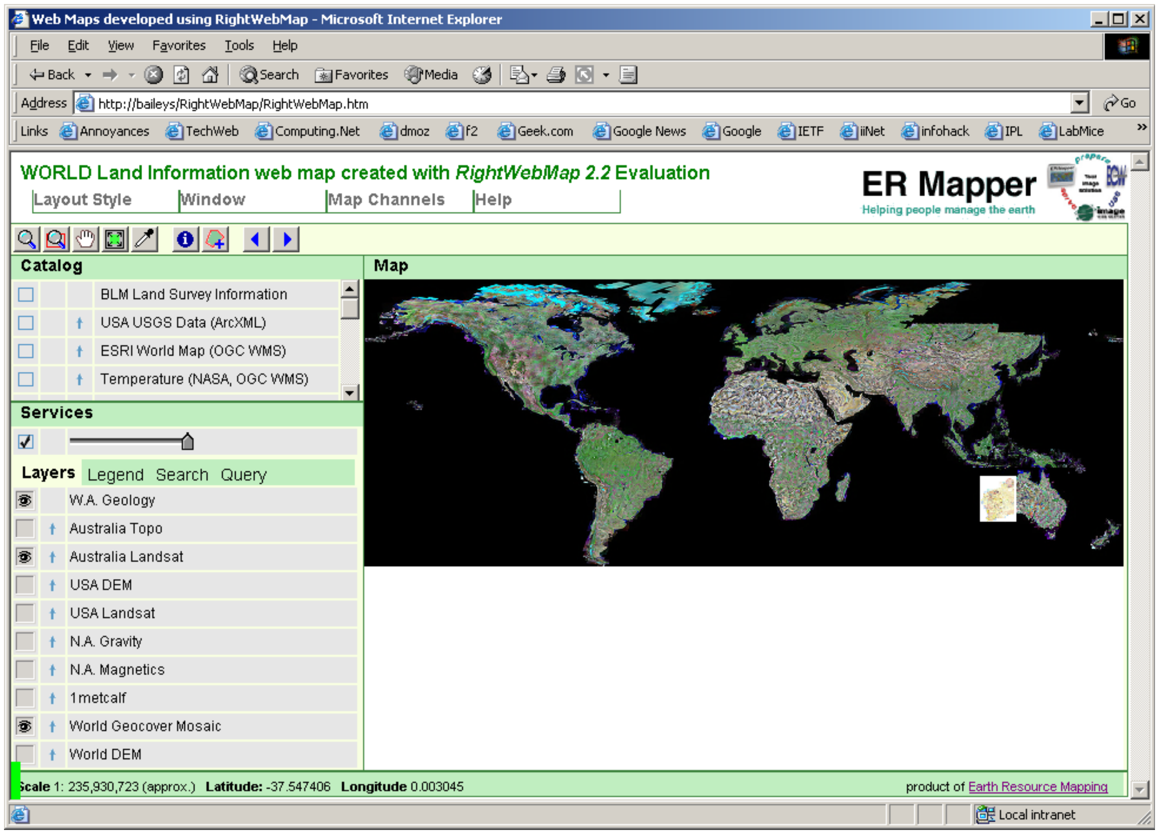
Using RightWebMap - an Overview

Introduction

RightWebMap is a rapid development and integration product for map servers. If you need a geospatial web solution which integrates multiple web map services into a single integrated web application, RightWebMap is probably the right product to consider.

An example web map page

The following screenshot show a typical RightWebMap generated web page.


Web Maps developed using RightWebMap - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://baileys/RightWebMap/RightWebMap.htm
Go
Links Annoyances TechWeb Computing.Net dmoz f2 Geek.com Google News Google IETF iiNet infohack IPL LabMice
WORLD Land Information web map created with RightWebMap 2.2 Evaluation
Layout Style Window Map Channels Help
ER Mapper
Helping people manage the earth
Catalog
Map
BLM Land Survey Information
USA USGS Data (ArcXML)
ESRI World Map (OGC WMS)
Temperature (NASA, OGC WMS)
Services
Layers Legend Search Query
W.A. Geology
Australia Topo
Australia Landsat
USA DEM
USA Landsat
N.A. Gravity
N.A. Magnetics
1metcalf
World Geocover Mosaic
World DEM
Scale 1: 235,930,723 (approx.) Latitude: -37.547406 Longitude 0.003045
product of Earth Resource Mapping
Local intranet

Web page elements

A Catalog in which a number of map services are listed. Selecting these services will display a map image in the LayeredView map window as a "stack" of images.

Each Service in the catalog can be queried to display a layers list for that service in the Services pane.

A slider bar to set transparency for each service. The transparency value is retained by the layer once it has been set.

Buttons are easily added to or removed from the toolbar to present a tailored suite of functions.

All of the images in the map view are georeferenced such that a change of extents in the map view automatically requests new imagery from all of the active services and updates the image layers for the respective services.

Rapid development

RightWebMap web pages can be created by adding just a few lines of JavaScript to a web page. This has been achieved by dividing the RightWebMap application into two distinct components:

- **Viewer Client (RightWebMap.htm)** - A simple page that can easily be setup by anyone with basic HTML coding skills. They can add services and ECWP image layers, and structure the toolbar.
- **Integration Service (RightWebMap.js)** - This is a powerful object-oriented JavaScript code for server integration. It has built-in interfaces for the all supported map servers. This code can be expanded by client system programmers to add new service interfaces and to expand reports and queries for existing server interfaces.

Creating a map page using JavaScript

RightWebMap.htm is designed as a logical and easy-to-understand configuration tool. Rightwebmap.htm is used to add map services, images to an Image Web Server service, add desired toolbar buttons and customize titles, logos, and other elements of the web page.

The Rightwebmap.htm references the rightwebmap.js JavaScript file as an include file and this file in turn contains the detailed JavaScript functions that interface to the services and provide the functionality of the page including drawing the maps to the LayeredView control.

The following examples illustrate the ease with which web map services can be added to the RightWebMap application:

Adding an OGC WMS service

With the single line of code below an OGC WMS service is added as a map service to the RightWebMap catalog of available services:

```
svc = catalog.addService("ESRI World Map (OGC
WMS)","wms","www.geographynetwork.com/servlet/
com.esri.wms.Esrimap","?ServiceName=ESRI_World",75,false);
```

Adding an Image Web Server ECWP service

This single line of code adds an ECWP (Image Web Server) service to the web map application's catalog:

svc = catalog.addService("Images (ECWP)","ecwp",0,0,100,true);

The following code adds an image called 'USA DEM' to the Layers list for the Images (ECWP) service and then sets the extents and projection type of this image using layer.setExtents() and layer.setXor methods.

```
layer = svc.addLayer("USA DEM    ","usadem    ","ecwp://www.earthetc.com/
images/usa/gs_dem.ecw                        ",false,false);
          layer.setExtents(51,-127,23, -64); layer.setXor("wgs84,geodetic");
```

Adding buttons to the Toolbar

The line of code below adds a zoom button to the toolbar of the RightWebMap application:

```
rTool.addTool("Zoom  ","zoom ",true,"mapZoom");
```

RightWebMap issues

Only Geodetic version of ArcXML and OGC WMS services are allowed at this point in time.

All of the ECWP images in the **Map** window must be in the same projection. Therefore, if a different image type is selected, it will unload the non-compatible images before loading the new image. This is controlled by code in RightWebMap_demo.htm 'layer.setXor("wgs84,geodetic"); ' which defines the datum and projection for the image and instructs RightWebMap to set ExclusiveOr so only images of the same definition can be on at the same time as this image.

RightWebMap uses the ECW plug-in Layered View control to display the maps. Currently, this control must have at least one ECW image layer to work, so a blank image is loaded and managed in the background. Future versions of the plug-in will remove this restriction.

3

Protocols

Common protocols supported by RightWebMap

RightWebMap has built-in support for the following protocols which are in wide use in the geospatial industry and which allow you to integrate the majority of GIS web map servers that are in use today. Added support for additional map servers in RightWebMap can be easily implemented. RightWebMap also allows easy integration of your map servers with Image Web Server to serve raster imagery.

Open GIS Consortium Web Map Service (OGC WMS) protocol

The Open GIS Consortium (OGC) is a world wide group of member companies who provide guidance and develop specifications for the GeoSpatial industry. The main site for the OGC is *http://www.opengis.org/*.

The OGC has evolved a standard for Client/Server interaction for delivery of geospatial information known as Web Map Service or WMS. The advantage of this Open standard is that it provides uniform interface rules which can be used by any vendor. Information for this interface can currently be found at *http://www.opengis.org/docs/01-022r1.pdf* .

The OGC Standards document defines a Web Map Service as follows:

“A Web Map Service produces maps of georeferenced data. We define a "map" as a visual representation of geodata; a map is not the data itself. These maps are generally rendered in a pictorial format such as PNG, GIF or JPEG, or occasionally as vector-based graphical elements in Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats. This specification standardizes the way in which maps are requested by clients and the way that servers describe their data holdings.”

This document defines three operations for version 1.1.1 of the Web Map Service Implementation Specification, the first two of which are required of every WMS:

- **GetCapabilities** (required): Obtain service-level metadata, which is a machine-readable (and human-readable) description of the information content and acceptable request parameters.
- **GetMap** (required): Obtain a map image whose geospatial and dimensional parameters are well-defined.
- **GetFeatureInfo** (optional): Ask for information about particular features shown on a map.

GetCapabilities

GetCapabilities allows a map client to interrogate a Web Map Service to determine what map layers are available for retrieval and display to the user. The results of a GetCapabilities request is returned to the client in an XML (eXtensible Markup Language) file which can be processed by the client software to extract the information required to make a GetMap request to access the information in a graphical image file.

WMS GetCapabilities Example:

A GetCapabilities request for an Image Web Server WMS Service would look like the following:

http://www.earthetc.com/ecwp/ecw_wms.dll?request=GetCapabilities&service=wms

This returns a list of all the images available, in XML format.

GetMap

GetMap sends a standard HTTP request from the client application to the Web Map Service to retrieve an image subset in a specified format. The formats supported by the GetMap specification are JPEG, PNG and GIF. The Image Web Server WMS Service supports the JPEG and the PNG formats. The image files are returned to the client if the request is valid, otherwise an error message is returned in XML format.

WMS GetMap Example:

A request for a "map" would look like this:

http://www.earthetc.com/ecwp/ecw_wms.dll?request=getmap&version=1.1.1&layers=/images/usa/sandiego3i.ecw&styles=none&srs=epsg:4326&bbox=-117.151317,32.706344,-117.149406,32.704725&width=200&height=200&format=image/png

Request Parameters

The following tables list the mandatory and optional parameters which can be used with GetCapabilities and GetMap.

GetCapabilities

Request parameter	Required/ Optional	Description
VERSION=version	Optional	Request version
SERVICE=WMS	Required	Service type
REQUEST=GetCapabilities	Required	Request name
UPDATESEQUENCE=string	Optional	Sequence number or string for cache control

GetMap

Request Parameter	Required/ Optional	Description
VERSION=version	Required	Request version
REQUEST=GetMap	Required	Request name
LAYERS=layer_list	Required	Comma-separated list of one or more map layers *
STYLES	Required	Comma-separated list of one rendering style per requested layer *
SRS=namespace:identifier	Required	Spatial reference system
BBOX=minx,miny,maxx, maxy	Required	Bounding box corners (lower left, upper right) in SRS units
WIDTH=output_width	Required	Width in pixels of map picture
HEIGHT=output_height	Required	Height in pixels of map picture
FORMAT=output_format	Required	Output format of map
TRANSPARENT=TRUE| FALSE	Optional	Background transparency of map (default-=FALSE)
BGCOLOR=color_value	Optional	Hexadecimal red-green-blue color value for the background color (default=0xFFFFFF)
EXCEPTIONS=exception_format	Optional	The format in which exceptions are to be reported by the WMS (default=SE_XML)

* **Exceptions from the standard:** A GetMap request may only request one layer at a time. Each layer is in fact a separate ECW file and cannot be blended into one image on the server side.

For more information on the GetCapabilities and GetMap request parameters see the OGC WMS specification: *http://www.opengis.org/docs/01-068r2.pdf.*

The Open GIS Consortium (OGC) has developed a standard for Client/Server interaction for delivery of Geospatial information known as Web Map Service or WMS. This protocol has the advantage that it is an Open standard designed to provide a uniform interface specification across a range of vendors. Currently the supported features provide the GetCapabilities and the GetMap requests in accordance to the Version 1.1.1 of the Web Map Service Implementation Specification. This new capability allows vendors of image data to take advantage of the efficient ECW compressed storage format while deploying an OGC compatible web service.

Adding ECW Images to the ArcXML Service

You can specify on a directory basis what ECW images you want to be made available from your Image Web Server for access via the ArcXML protocol. This is done using the Image Web Server Console. See the previous section on the WMS service for details.

Note: The process of exposing images to the ArcXML service is exactly the same as that of adding images to the WMS service since the images added become available to both services.

The images that have been made available are reported during an ArcXML query and will be shown in the layers list for applications using the format.

Adding ECW Images to the WMS Service

By specifying a directory containing ECW images you can make these images available from your Image Web Server for access via the WMS protocol. This is done using the Image Web Server console.

Enhanced Compression Wavelet Protocol (ECWP)

The Image Web Server provides a highly efficient imaging archival service which stores images in the ECW compression format, allowing compression ratios up to 1:20 for a 95% reduction in image storage size and costs. This patented compression format is designed to economically deliver images to the mapping clients in streaming format via an ECWP protocol allowing web-enabled applications like browsers to efficiently access very large images even on slow dial-up Internet connections.

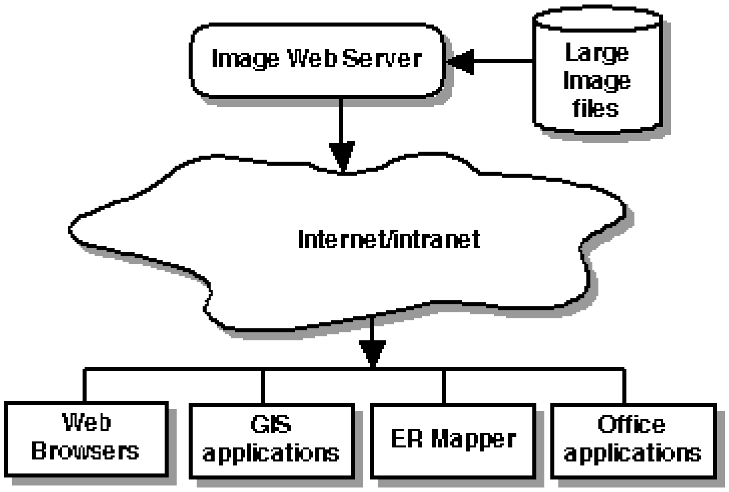


Image Web Server provides a very powerful "streaming imagery" solution known as Enhanced Compression Wavelet Protocol (ECWP) which is layered over standard HTTP and provides image data on demand from compressed ECW image files. To enable web browsers to use this protocol requires a one-time download (600Kbytes) of a free ECW plug-in. Alternately a Java Applet is available which does not require the plug-in but does require the Sun Java runtime (JRE) support. Free plug-ins are also available for a large range of 3rd party applications such as ArcGIS and ArcView and are available from *http://www.ermapper.com/ downloads/*.

The ECWP protocol is recommended for heavy use, enterprise systems since it provides the fastest response to the client applications and reduces the network load dramatically compared to shipping images extraction files in the form of JPEG, GIF or PNG image files.

The advantage of the ECWP protocol is that it provides significant performance enhancements over other protocols because it uses streaming technology with local caching and real-time pan and zoom. This can provide much faster response and significantly reduced network traffic.

The only requirement of using the ECWP format for any application is that it requires the user to download and install enabling software in the form of a plug-in or in the case of the Java applet, a Java runtime environment (JRE).

The client application caches the image blocks locally (for a session) so subsequent data requests over the same area do not require resending of these data blocks from the server. This results in a significant saving on network load for high end image users. This also provides near real-time pan and zoom compared to few seconds delay encountered from GIS Servers that request a new image every time the map extents are changed.

4

Developer's integration tutorial

Using RightWebMap versus I-Wizard

Two products are offered by ER Mapper to integrate ArcIMS generated web maps with imagery: RightWebMap and the I-Wizard. Both products may be downloaded from the www.ermapper.com downloads page.

ArcIMS based web sites can choose to use either RightWebMap or I-Wizard; both offer different advantages.

Integration of Spatial Services

Enterprise-scale solutions in the spatial sector are becoming ever more demanding with managers needing to provide their clients with an integration of data services, often from a variety of vendors and stored in different formats. There is a requirement to provide integrated applications in a web-enabled framework for use in both intranets and for deployment over the Internet.

Standards committees like the OGC are making progress in evolving Open Interface standards, designed to smooth the way for integration of multi-vendor solutions which can encompass both legacy data sets as well as future data formats. However, many companies have large legacy data resources already in place that may not be available or fully deployable from an OGC standard interface.

ER Mapper has recognised the need for tools to efficiently integrate data services and offer two very powerful tools to assist spatial data users to seamlessly integrate GIS data with Image Web Server data in a web browser application. These are:

- **I-Wizard** - a free application that integrates Image Web Server ECWP support with ESRI® ArcIMS® Arc Designer generated web sites.
- **RightWebMap** - a high-end rapid development application that integrates multiple data services in a LayeredView presentation within a web browser application. RightWebMap is a licensed product which is sold on a per server basis. An evaluation copy is installed with Image Web Server.

Integration with RightWebMap

Apart from integrating map layers from many different GIS servers, RightWebMap also provides more functionality than I-Wizard and therefore provides the preferred integration solution for most situations.

Use RightWebMap instead of the I-Wizard when:

- When you want to integrate multiple servers into the same map view
- You want to integrate servers that are not ArcIMS based into your web maps. RightWebMap can integrate ArcIMS, OGC/WMS, MapXtreme, MapServer, MapGuide, WebMap and many other server sources into a single map view.
- You want more advanced looking web map look and feel
- You want to customize the style for web maps
- You want advanced imagery display and transparency
- You need to quickly develop and integrate advanced web maps

Use I-Wizard to integrate with ArcDesigner created web pages

The I-Wizard integrated with ArcDesigner created pages.

It adds the ECW Control and associated JavaScript to allow ArcDesigner pages quickly access imagery. Advantages include:

- Tightly integrated with existing ArcDesigner created pages
- Speeds imagery access for ArcIMS based web map sites

However, unless using existing ArcDesigner based web maps is a critical requirement, RightWebMap is likely to be your preferred solution as it adds many other capabilities.

Use RightWebMap for rapid development and multiple servers

RightWebMap integrates map layers from many types of servers including from ArcIMS servers.

Use RightWebMap instead of the I-Wizard when:

- When you want to integrate multiple servers into the same map view
- You want to integrate servers that are not ArcIMS based into your web maps. RightWebMap can integrate ArcIMS, OGC/WMS, MapXtreme, MapServer, MapGuide, WebMap and many other server sources into a single map view.
- You want more advanced looking web map look and feel
- You want to customize the style for web maps
- You want advanced imagery display and transparency
- You need to quickly develop and integrate advanced web mapsBuilding a RightWebMap authored web map page

Creation of a RightWebMap authored web map page requires only a few lines of instructions.

Integration Tutorial

Hopefully you are now impressed by the power of the RightWebMap tool to integrate web maps from a number of different servers in a single layered view display.

With equal ease, you can customize the RightWebMap_demo.htm page (or create a new one from scratch) to display the services that you choose.

The reason that configuring RightWebMap is so easy to do is because much of the code-intensive tasks have already been built into the classes which are resident in the RightWebMap.js include file.

As long as the interface for your particular service type (the main server types are there) is present, you can simply add a few lines of code to the RightWebMap.htm file to invoke these classes.

In the following section we go through the code section-by-section with an explanation of how the services and images are added into the .htm page.

Adding an ECWP Service to the Catalog

This single line adds the ECWP service to the **Catalog**:

```
svc = catalog.addService("Image Web Server Images (ECWP)",
➥"ecwp",0,0,100,true);
```

Its components are:

Name	Name shown in Catalog 'Image Web Server Images (ECWP)'.
InterfaceName	Defines type of service in this case 'ecwp'.

Host	Not required for ECWP services.
Service	Not required for ECWP services.
Percent	Transparency: 100 is no transparency (opaque), 0 is fully transparent.
Active	Boolean true indicates start with service on (checked).

Adding an image to the ECWP service

The next line of code adds an image to the service we created above. This image will be displayed in the Layers List of the Service Pane when the Service name is selected in the Catalog.

```
layer = svc.addLayer("USA Landsat - EarthEtc","usalandsat",
†"ecwp://www.earthetc.com/images/usa/gs_landsat_lowres.ecw",
†false,false);
```

Its components are:

name	Name displayed in Layers list: 'USA Landsat - EarthEtc'.
id	Name used by JavaScript; in this case 'usalandsat'.
url	The full ECWP URL for the image.
visible	Controls if image is on (checked) or off (false = not checked).
doQuery	Can you perform a query on the layer?. False = no.

Setting the extents for the image

RightWebMap uses extents to automatically turn the image off if the map view is outside the defined extents. This is to minimize the network traffic and increase client performance, as otherwise redundant imagery could persist and use up resources.

```
layer.setExtents(51,-127,23, -64);
```

Its components are:

tlLatitude	51
tlLongitude	-127
brLatitude	23
brLongitude	-64

Setting the projection for the Image

The next line sets the projection for the image. The LayeredView control (which displays the imagery in the Map window) can only handle a single projection system at a time. The **setXor** method is used to define the datum and projection. This information is used by RightWebMap to automatically unload all non-compatible images from the 'stack' before this image is displayed.

```
layer.setXor("wgs84,geodetic");
```

Its components are:

Datum	wgs84
Projection	geodetic

Adding additional images to the ECWP Service

The code below adds two images from a local Image Web Server and a third one from the www.earthetc.com web server. Note that:

- All of the images are in the same datum/projection so they can all be shown in the view at the same time.
- The LocalServer variable is used to set the path for individual machines.
- Only the last image is set to be on at startup - i.e., the visible parameter is true (checked).

```
     layer = svc.addLayer("USA Demo Image - Local Image Web
Server","usa_demo","ecwp://" + LocalServer + "/sampleiws/images/geodetic/
USADemo.ecw",false,false);
         layer.setExtents(66.14, -155.358, 7.175, -44.63);
layer.setXor("wgs84,geodetic");
     layer = svc.addLayer("World GeoDemo - Local Image Web
Server","world_geodemo","ecwp://" + LocalServer + "/sampleiws/images/
geodetic/WorldGeoDemo.ecw",false,false);
         layer.setExtents(90, -180, -90, 180);
layer.setXor("wgs84,geodetic");
         layer = svc.addLayer("World Geocover Mosaic - EarthEtc"
,"World_Geocover_Mosaic" ,"ecwp://www.earthetc.com/images/geodetic/world/
landsat742.ecw"
              ,true,false);
         layer.setExtents(90,-180,-90,180);   layer.setXor("wgs84,geodetic");
```

Adding the local OGC WMS Service

The code below adds the WMS service that you created in your Image Web Server at the beginning of the tutorial.

```
     svc = catalog.addService("Local OGC WMS Server","wms",LocalServer + "/
ecwp/ecw_wms.dll?","wms",100,false);
```

Its components are:

Name	Name shown in Catalog 'Local OGC WMS Server'.
InterfaceName	Defines type of service in this case 'wms'.

Host	URL to WMS server - your machine in this case.
Service	Launches the server - "/ecwp/ecw_wms.dll?.
Percent	Transparency: 100% means no transparency (opaque).
Active	Boolean false indicates start with service off.

Setting a ColorMask for transparency on the WMS JPEG Image

The WMS interface in RightWebMap is set to request a JPEG image since they are smaller in size compared to 24-bit PNG images (the alternate option).

This causes a problem when setting transparency as it is difficult to remove colors from JPEG image map layers. This is because a single original map color may be converted to many colors during JPEG compression.

Where a server returns a JPEG image, a `colorMask` can be specified for that service. This is used as a mask during map color transparency calculations. The mask is specified as a 24-bit hexadecimal number. It should be set for the service before any color operations, such as **hideColor**, are performed.

For example:

```
eviewSvc.colorMask = 0xf0f0f0;
```

This is the mask that was used in the demo page:

```
svc.colorMask = 0xf0f0f0;
```

Hiding Colors on the WMS Service

For images served by the Image Web Server WMS Service, it is necessary to set transparency on fill color (white) that surrounds an image when the view is larger than the full image extents, as otherwise images underneath will be obscured. This is achieved by using the `hideColors` method. It allows any individual color to be specified in RGB format and made fully transparent for the image. Multiple colors can be specified, separated by commas, as in this example:

```
svc.hideColors(["#ffffff","#F8F6C4"]);
```

The line below makes the white fill background fully transparent on the images from your local WMS server.

```
svc.hideColors(["#ffffff"]);
```

Adding your local ArcXML Services

The code below adds the ArcXML service that you created at the beginning of the tutorial for the image USADemo.ecw.

svc = catalog.addService("Local ArcXML Server - USA","arcxml",
✝LocalServer,"/sampleiws/images/geodetic/USADemo.ecw",100,false);

Its components are:

Name	Name shown in Catalog: 'Local ArcXML Server - USA'.
InterfaceName	Defines type of service in this case 'arcxml'.
Host	URL to ArcXML server - your machine in this case.
Service	Path from root - '/sampleiws/images/geodetic/USADemo.ecw'.
Percent	Transparency: 100% means no transparency (opaque).
Active	Boolean false indicates start with service off.

Note that the ArcXML service is different from the WMS example above in that each image is set up as a separate service with the image as a single layer in that service. The WMS has a single Service with all of the images reported as layers during a GetCapabilities request.

The following code hides the white fill background similar to the WMS Service described in detail above.

svc.hideColors(["#ffffff"]);

Adding a second image to the ArcXML service

The code below adds a second image - WorldGeoDemo.ecw - to the ArcXML service and hides the white background color using the method described earlier.

svc	= catalog.addService("Local ArcXML Server - World","arcxml",LocalServer,"/sampleiws/images/geodetic/WorldGeoDemo.ecw",100,false);
	svc.hideColors(["#ffffff"]);

Adding an External ArcXML service to the catalog

The code below will add an ArcXML service available over the Internet to the catalog of the demo page. The components of this code should now be familiar to you so it will not be described in detail.

```
svc = catalog.addService("BLM Land Survey
(ArcXML)","arcxml","www.lsi.blm.gov","BLM_LSIS",75,false);
     svc.hideColors(["#ffffff","#F8F6C4"]);
```

Adding an External OGC WMS service to the catalog

The code below will add an ESRI WMS server to your catalog. Note that we have started the service transparency slider at 75% instead of 100 to help show the imagery underneath and two fill colors have been identified and made fully transparent using the `hideColors` method.

```
svc = catalog.addService("ESRI World Map (OGC
WMS)","wms","www.geographynetwork.com/servlet/
com.esri.wms.Esrimap","?ServiceName=ESRI_World",75,false);
     svc.hideColors(["#ffffff","#ccffff"]);
```

What you learned...

After completing this tutorial, you know how to perform the following tasks with Image Web Server and RightWebMap:

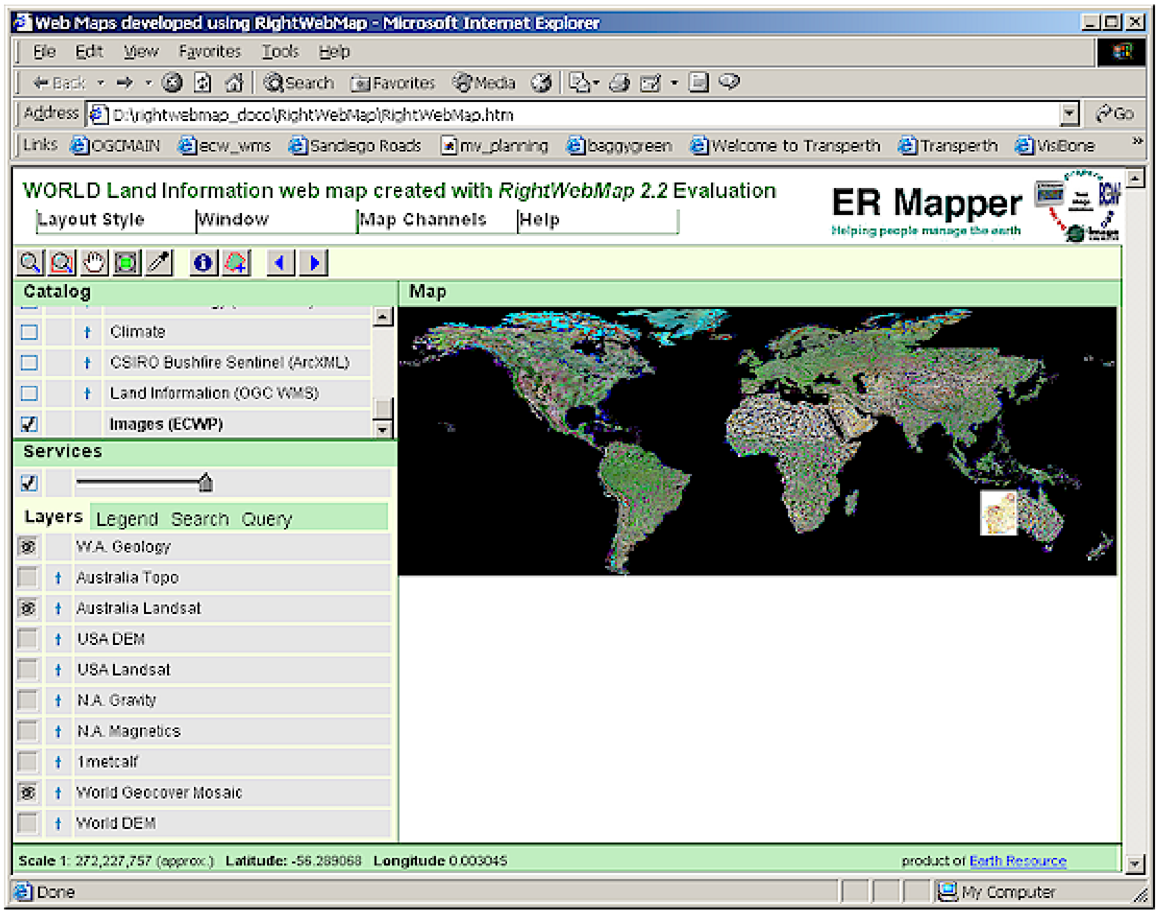
- Integrate multiple web map servers into a single view.
- Integrate OGC/WMS, ECWP (Image Web Server) and ArcIMS (ArcXML) servers.
- Set up interactive layering, stacking and transparency between map layers from different servers.

You should have a good grasp of how to design a RightWebMap application for your own servers and your own needs. You are encouraged to make a copy of the RightWebMap_demo.htm file and use that to experiment with.

Creating a map page using JavaScript

A page can be easily created by including the RightWebMap JavaScript and CSS files, and defining the map objects.

An example web map page


Web Maps developed using RightWebMap - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links OGCMAIN ecw_wms Sandiego Roads mv_planning baggygreen Welcome to Transperth Transperth VisBone
WORLD Land Information web map created with RightWebMap 2.2 Evaluation
Layout Style Window Map Channels Help
ER Mapper
Helping people manage the earth
Catalog
Climate
CSIRO Bushfire Sentinel (ArcXML)
Land Information (OGC WMS)
Images (ECWP)
Services
Layers Legend Search Query
W.A. Geology
Australia Topo
Australia Landsat
USA DEM
USA Landsat
N.A. Gravity
N.A. Magnetics
1metcalf
World Geocover Mosaic
World DEM
Map
product of Earth Resource
Done
My Computer

Address D:\rightwebmap_docs\RightWebMap\RightWebMap.htm

Scale 1: 272,227,757 (approx.) Latitude: -56.289068 Longitude 0.003045

Analysis of a web map Configuration Page (RightWebMap.htm)

RightWebMap has been designed with a very logical and easy to understand **Configuration Page (RightWebMap.htm)**, which allows designers who are not expert in DHTML to easily configure a web map application. **Rightwebmap.htm** is used to add the required map services, add images to an Image Web Server service, add desired toolbar buttons and customize titles, logos, etc. The **Rightwebmap.htm** references the rightwebmap.js javascript file as an include file and this file contains the detailed javascript functions that interface to the services that have been defined, handle the drawing of the maps to the LayeredView control incorporated into the RightWebMap web map page, perform queries etc.

The following is a detailed breakdown of the RightWebMap.htm page provided in the evaluation installation of RightWebMap and a discussion of how to configure this page to create your own application.

Include Common Plug-in Checking Functions

Code:

```
<!-- include common plugin check files -->
<script src="/ecwplugins/lib/scripts/NCSCheck.js" type="text/javascript"></
script>
<script src="/ecwplugins/lib/scripts/NCSCheckActiveX.vbs" type="text/
vbscript"></script>
<script src="/ecwplugins/lib/scripts/NCSConstants.js" type="text/
javascript"></script>
<script src="/ecwplugins/lib/scripts/NCSCreateView.js" type="text/
javascript"></script>
```

Notes:

The common javascript functions, packaged with image webserver, need to be included in the RightWebMap.htm page. These common functions are used within RightWebMap to detect browser and plugin versions. Where necessary, users are prompted to follow a particular upgrade path.

Checking for Javascript version type

Code:

```
<script language='JavaScript'> var _version = 1.0 </script>
<script language='JavaScript1.1'> _version = 1.1 </script>
<script language='JavaScript1.2'> _version = 1.2 </script>
<script language='JavaScript1.3'> _version = 1.3 </script>
<script language='JavaScript'>
   if( _version < 1.2 ) {
       location.replace("upgrade_browser.htm");
       alert("This application needs a more recent version of your web browser
to run correctly.");
   }
```

Notes:

The RightWebMap applications require a minimum JavaScript version of 1.2. If an older version is encountered the user is directed to the ‘upgrade_browser.htm’ page which advises the user what versions of browsers are required. You can modify this page to present a message that you feel is appropriate.

Adding in the CSS Reference

Code:

```
<link rel="stylesheet" type="text/css" href="RightWebMap.css">
```

Notes:

The RightWebMap application is provided with a stylesheet file called **RightWebMap.css.** This style sheet may be modified by developers to create their own page colors and styles.

You will note that the RightWebMap application can operate in either a Frames presentation or a Windows presentation. There are components of the stylesheet that are specific to these modes.

Adjusting the Catalog Pane Height

Code:

```
var nPaneHeight = 125;// controls the catalog pane height
```

Notes:

Depending on the number of map services that you specify in your web map catalog of services, you will need to adjust this variable to provide the optimal presentation.

Initializing the Application Object

Code:

```
// Initiate the application
var rApp = new RApp();
```

Notes:

This code is used to create a new application object and should be included as is. See the **Internal object design** for an explanation of how objects work in RightWebMap.

Starting the Application

Code:

```
function startup() {
   rApp.build();
   // reset the exents on the image...
   var iTime = setTimeout(setStartingExtents, 10)
```

```
}
function setStartingExtents()
   {
   var map = document[rApp.rMaps[0].uid];
   map.SetExtentsAll();
```

Notes:

This code is used to perform a build of the application whereby the various window objects, toolbar objects, map objects etc., which have been defined, are set up. The rApp.build() code should always be included as shown.

The function setStartingExtents() is called on a timer (to allow the objects to be fully built first) and is used to establish the starting extents for the map view on application startup. In this case the code **map.SetExtentsAll();** defines that the map display should start at the full extents of the largest image which has been selected for display on application startup. If a specific set of map extents are desired then this code can be changed to **map.SetExtents(tlx,tly,brx,bry);** where:

tlx:	**TopLeftWorldCoordinateX**
tly:	**TopLeftWorldCoordinateY**
brx:	**BottomRightWorldCoordinateX**
bry:	**BottomRightWorldCoordinateY**

These coordinates should be specified in the coordinate system of the image being added.

Defining the default map channel

Code:

```
if( rApp.maps == "" )
   rApp.maps = "world";
```

Notes:

The application can set this parameter to control what group of maps to display. This code initializes the display to be the group we will define as the "world" group. The Channels menu is used to switch this later to other options such as "USA" or "Australia" and javascript presents different services controlled by the setting of this parameter.

```
var loadAustralia = (rApp.maps != "usa" ? true : false);
var loadUSA = (rApp.maps == "usa" || rApp.maps == "world" ? true : false);
```

Controlling the Channels

Code:

```
var channels = new Array();
   channels[rApp.maps] = true;

varloadAustralia = (rApp.maps != "usa" ? true : false);
varloadUSA = (rApp.maps == "usa" || rApp.maps == "world" ? true : false);
```

Notes:

This code sets boolean variables based on the setting of the map group parameter; these variables will be used later in the page to control the services and image layers presented to the user.

Setting the Title for the Application

Method:

```
SetTitle(title as string)
```

Code:

```
rApp.setTitle(rApp.maps.toUpperCase() + " Land Information web map created with
<i>RightWebMap " + rApp.version + "</i>");
```

Notes:

This code converts the name of the current map group (rApp.maps) to uppercase and concatentates it with a text message and the version number of the RightWebMap application.

Setting up the Toolbar Window

Method:

```
AddWindow(type as string, name as string)
RToolbar(window as object)
AddObject(toolbar as object)
```

Code:

```
var toolWin = rApp.addWindow("toolbar","Toolbar");
var oStatus = new RToolbar(toolWin)
var rTool = toolWin.addObject(oStatus);
```

Notes:

Adds a new window object to the application object designed to hold the toolbar we will create.

Then a new toolbar object is created and added as a child to the toolWin object create previously.

Adding Toolbar Buttons to the Toolbar

Method:

```
AddTool(name of button as string, name of button image as string, sticky* as
boolean, name of function to call as string
```

Code:

```
rTool.addTool("Zoom","zoom",true,"mapZoom");
rTool.addTool("Zoom Box","zoombox",true,"mapZoomBox");
rTool.addTool("Pan","pan",true,"mapPan");
rTool.addTool("Zoom All","setExtentsAll",false,"mapZoomAll");
rTool.addTool("Hide colors","transparent",true,"mapTransparent");
rTool.addSpace("Column");
rTool.addTool("Query a point" ,"query_point",true,"mapQueryPoint");
rTool.addTool("Query a polygon","query_polygon",true,"mapQueryPolygon");
```

Notes:

This code above creates a toolbar by adding a number of tools (buttons) to the toolbar object. The parameters that get specified when we add a button are defined as follows.

*Sticky indicates that when pressed, the button will appear depressed until the application enters a different mode.

For Example:

```
rTool.addTool("Zoom", "zoom", true, "mapZoom");
```

`rTool`	the name of the toolbar object we created above
`.addTool`	the method used to add tools (buttons) to the toolbar object
"Zoom"	the ToolTip that shows up if you hover over the button
"zoom"	the name used to access the buttons images from the /images folder. In this case there will be a 'zoom.gif' and a 'zoom_on.gif' for latched mode
true	is it a "sticky" tool – true is sticky
"mapZoom"	the internal pointer mode set when the button is clicked

In order for the mouse over effects for the tools buttons to work correctly, the image must contain at least two versions. "Normal" or "Inactive" as "nameofimage.gif" and active as "nameofimage_on.gif".

Setting up the Map Display Window

Code:

```
var mapWin  = rApp.addWindow("main","Map");
var map = mapWin.addObject(new RMap(mapWin));
```

Notes:

A new window object called "main" is added to the application object. The string "Map" is displayed in this window's banner. (Note that you can change the name displayed simply by changing this string value).

Then a new Map object is created for display within the window we created (is added as a child object to the mapWin object).

Setting up the Services Pane Display Window

Code:

```
var allWin= rApp.addWindow("pane","Catalog");
var catalog = allWin.addObject(new RCatalog(allWin));
catalog.attach(map);
```

Notes:

A new window object of type “pane” is added to the application object. The string “Catalog” is displayed in this window’s banner. (Note that you can change the name displayed simply by changing this string value).

A new catalog object is created and added to this window object.

Finally the catalog object is “attached” to the map object we created above to associate them together so active services in the catalog will display into the map-viewing window.

Setting up the Status Pane Display Window

Code:

```
var statusWin= rApp.addWindow("status","Status");
var rStatus = statusWin.addObject(new RStatus(statusWin));
map.attach(rStatus);
```

Notes:

A new window object of type “status” is added to the application object. The string “Status” is displayed in this window’s banner (unless overwritten by information). You can change the name displayed simply by changing this string value.

Then a new status object is created and added to this window object.

Finally the status object is “attached” to the map object we created above to associate them together so map activities can be reflected into the status window.

Adding logo(s) to your web map page

Method:

```
AddLogo(HTML as string)
rApp.addLogo("<a href=http://www.ermapper.com><img src='images/logoleft.gif'
border='0'></a>");
rApp.addLogo("<a href=http://www.earthetc.com><img src='images/logoright.gif'
border='0'></a>");
```

Code:

```
rApp.addLogo("<a href=http://www.ermapper.com><img src='images/logoleft.gif'
border='0'></a>");
rApp.addLogo("<a href=http://www.earthetc.com><img src='images/logoright.gif'
border='0'></a>");
```

Notes:

Logos can be added to your page using the addLogo method of the rApp object. Follow normal html coding in defining the string to insert into this call.

Adding an ECWP Service to your web map page

Method:

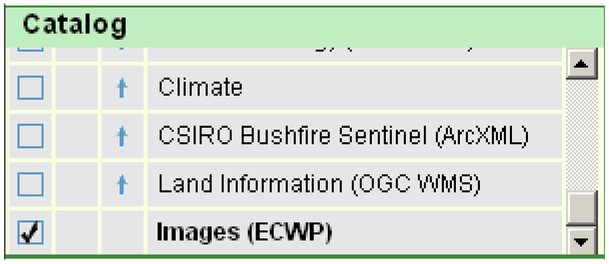
addService(label as string, access method as string, service url as variant, service name as variant, percent transparent as integer, service active as boolean)

Code:

```
var svc;
svc = catalog.addService("Images (ECWP)","ecwp",0,0,100,true);
```

Notes:

We use the .addService method of the catalog object to add a new service to the catalog which will show up in the Catalog pane identified by the name string which is "Images (ECWP)" (see bolded item below).



Note: The services show up in reverse order that they appear in the code (eg the first inserted into the code will be the last in the catalog list)

The "ecwp" stands for Enhanced Compression Wavelet Protocol and indicates that this map service is an Image Web Server. When the selection checkbox is clicked the javascript code uses this parameter to direct the program to the ECWP interface handler.

The three values following (0,0,100,true) are:

0	host url is not necessary for ECWP service. URL's are given when adding layers for this service. **All other services require this value.**
0	Service name is not necessary for ECWP service. **All other services require this value.** **100**sets the transparency for this map service from 0

	fully transparent to 100 fully opaque **true**defines that the service is selected on start up of the web map page

Adding an ECWP Layers (ECW Images) to your web map page

Method:

```
AddLayer(label as string, internal id as string, url as string, visible as
boolean, query as boolean )
SetExtents(as double*, as double, as double, as double)
SetXor(datum and probject as string)
```

* Defined as double, although, as javascript doesn’t have strongly defined types, you can enter integer / float values.

Code:

```
   var layer;

   // world - huge
   layer = svc.addLayer("World Geocover Mosaic","World_Geocover_Mosaic
","ecwp://www.earthetc.com/images/geodetic/world/landsat742.ecw"
          ,true,false);
   layer.setExtents(90,-180,-90,180);
   layer.setXor("wgs84,geodetic");
```

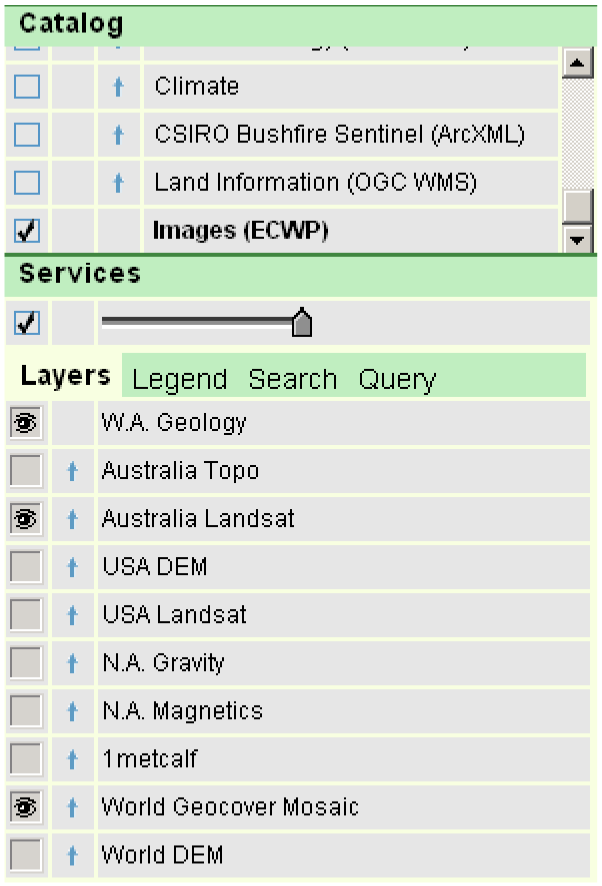
Notes:

The addLayer method is used to add new layers for the ecwp service we defined above (referenced by ‘svc’) and the layer object assigned to the variable declared earlier called ‘layer’. The components which are passed to the method are as follows:

For example:

```
layer = svc.addLayer("USA DEM","usadem","ecwp://www.earthetc.com/images/usa/
gs_dem.ecw ",false,false);
```

“USA DEM”	the name displayed in the layers list for this service. See the layers list for the images(ECWP) service below



"usadem"	internal name defined for this layer

"ecwp://www.earthetc.com/images/usa/gs_dem.ecw"the path to the image

false	This Boolean parameter controls whether the image layer is turned on (loaded to the LayerView Control when the application is started. Note that three images are set to load on startup; W.A. Geology, Australia Landsat, and World Geocover Mosaic. If you check the addLayer code for these images you will see this parameter is defined as true (see example below).

layer = svc.addLayer("World Geocover Mosaic" ,"World_Geocover_Mosaic","ecwp://www.earthetc.com/images/geodetic/world/landsat742.ecw"

,**true**,false);

false	This Boolean parameter defines whether the image should always be on. Normally this would be set to false except for blank images that may be defined as always on.

layer.setExtents(51,-127,23, -64);this code uses the setExtents method of the layer object to define the top left and bottom right extents for the map. The extents are set in geodetic coordinates as follows (tlLatitude, tlLongitude, brLatitude, brLongitude). Note: The extents must be defined in latitude and longitude even if the image's is not geodetic. These coordinates are used to hide/show the image in the Services Pane. If the extents of the map are inside the coordinates specified the image will be shown in the list, if outside the image will not be shown in the list.

layer.setXor("wgs84,geodetic"); this code uses the setXor method of the layer object to define the coordinate system for the image layer. It is defined in order to allow the software to determine which images are compatible for display since only images in the same datum/projection can be displayed at one time in the LayeredView control. If an image from a non-compatible coordinate system is selected all images currently selected will be unselected and the new image will load to the map view window. If the view is zoomed outside the extents of the new image it will be resized to the bounding extents of the new image. If the map view is zoomed within the extents of the new image these extents will be retained.

Adding an OGC WMS Service to your web map page

Method:

```
hideColors(color as string array);
```

Code:

```
svc = catalog.addService("Land Information (OGC
WMS)","wms","atlas.walis.wa.gov.au/servlet/
com.esri.wms.Esrimap?ServiceName=","WA_Atlas_GN",100,false);
   svc.hideColors(["#ffffff","#F8F6C4"]);
```

Notes:

The code for adding an OGC WMS service to the catalog is essentially the same as adding the ecwp service described above.

For GIS servers which provide a png or a gif image file fill colours can automatically be made fully transparent using the hideColors method of the service object as shown above. An array of colors can be specified in hex values.

It is not necessary to add individual layers for this service. The initiating call to the service asks for a list of layers. The list of layers is populated in RightWebMap automatically.

Add an ArcXML Server

Code:

```
svc= catalog.addService("USA USGS Data
(ArcXML)","arcxml","seamless.usgs.gov","ZipnShip",100,false);
svc.hideColors(["#000000"]);
```

Notes:

“ZipnShip” is the ArcIMS service we want to connect to.

Add a Mapserver

Notes:

The map service in this instance is the physical file location of the .map file. Layers for the MapServer need to be added manually.

```
svc = catalog.addService("USA (MapServer)","mapserver","www.earthetc.com/
mapserver/mapserv.exe","f:/wwwroot/html/ecwgis_geology_17/
geology.map",100,false);
   svc.hideColor("#000000",0,0,false);
   svc.addLayer("Hydrography","hydrography","",false,false);
   svc.addLayer("Geology Polygons","geologyp","",false,false);
   svc.addLayer("Fault lines","faults","",false,false);
   svc.addLayer("Counties","counties","",false,false);
   svc.addLayer("States","states","",true,false);
   svc.addLayer("Urban areas","urban","",false,false);
   svc.addLayer("Roads","roads","",true,false);
   svc.addLayer("Mine sites","mines","",false,false);
```

Adding other Map Services to your web map page

Throughout the page you will see a number of services being added to the catalog of available services. For instance in the USA section the following servers are added:

Code:

```
if( loadUSA ) {
svc = catalog.addService("USA (MapServer)","mapserver","www.earthetc.com/
mapserver/mapserv.exe","f:/wwwroot/html/ecwgis_geology_17/
geology.map",100,false);
   svc.hideColor("#000000",0,0,false);
   svc.addLayer("Hydrography","hydrography","",false,false);
   svc.addLayer("Geology Polygons","geologyp","",false,false);
   svc.addLayer("Fault lines","faults","",false,false);
   svc.addLayer("Counties","counties","",false,false);
   svc.addLayer("States","states","",true,false);
   svc.addLayer("Urban areas","urban","",false,false);
   svc.addLayer("Roads","roads","",true,false);
   svc.addLayer("Mine sites","mines","",false,false);


svc = catalog.addService("Life Mapper","wms","www.lifemapper.org/Services/
WMS","?service=WMS",100,false);
   svc.hideColor("#ffffff",0,0,false);
svc = catalog.addService("USA USGS (WMS)","wms","gisdata.usgs.net/servlet/
com.esri.wms.Esrimap?","",75,false);
   svc.hideColors(["#ffffff","#dbd79b","#009933"]);

svc = catalog.addService("Temperature (NASA, OGC
WMS)","wms","wmt.digitalearth.gov/cgi-bin/wmt.cgi?","",75,false);

svc = catalog.addService("ESRI World Map (OGC
WMS)","wms","www.geographynetwork.com/servlet/
com.esri.wms.Esrimap","?ServiceName=ESRI_World",75,false);
   svc.hideColors(["#ffffff","#ccffff"]);

svc= catalog.addService("USA USGS Data
(ArcXML)","arcxml","seamless.usgs.gov","ZipnShip",100,false);
svc.hideColors(["#000000"]);

svc = catalog.addService("BLM Land Survey
Information","arcxml","www.lsi.blm.gov","BLM_LSIS",100,false);
   svc.hideColors(["#ffffff"]);
}
```

Notes:

Use these examples as templates for adding your own services depending on what type of GIS server you wish to access. \

Adding drop-down menus to your web map page

Code:

```
var styleMenu = rApp.addMenu("Layout Style");
styleMenu.addItem("Windows",'javascript:rApp.loadPage("Win")' );
styleMenu.addItem("Frames",'javascript:rApp.loadPage("Frame")'  );

var windowMenu = rApp.addMenu("Window");
windowMenu.addWindowList();
```

Notes:

The addMenu method of the application object is used to create drop-down menus for your web page.

▪The string parameter within addMenu defines the title for the menu (eg "Layout Style").

The addItem method is used to add items to the dropdown menu list. Note that a javascript function call may be included to execute when an item is selected.

Note that the menu defined as "Window" populates it's menu list using the method addWindowList() which draws on the array of windows which have been created in the page.

Setting up the Body Tag Events for your page

Code:

```
<BODY onload='startup()' onResize='onWindowResize()' topmargin="2"
leftmargin="2" bgproperties="fixed">
```

Notes:

onload='startup() – the body onload event calls the ‘startup()’ function defined earlier in the page to build the application object and set the starting extents for the map view.

onResize='onWindowResize() – whenever the browser window is resized map viewing extents must be recalculated and new server map requests generated. This function call handles that based on the page resize event.

Initialising the application object

Code:

```
// Initiate the application
var rApp = new RApp();
```

Notes:

This JavaScript code is called on page loading to invoke the init method of the application object which sets pre-defined parameters.

4

System Developer Information

This section is an in-depth treatment of the system design and the JavaScript classes that are used in RightWebMap and will be of particular benefit to system developers who are involved in adding new service interfaces to a RightWebMap project.

Application developers who are customizing RightWebMap using existing functionality will benefit from this section in understanding the overall structure of the system and how they can modify the presentation elements of the product in the areas of style, window layout, toolbar implementation and adding and controlling the presentation of services and layers.

Internal object design

RightWebMap is built using JavaScript 1.2 (note that JavaScript is the standard web browser language, it is not Java despite having a similar language).

RightWebMap uses a object and method style of design. This means that a web page contains objects (such as windows, toolbars, and so forth) and each object has different methods (or functions) to control the object.

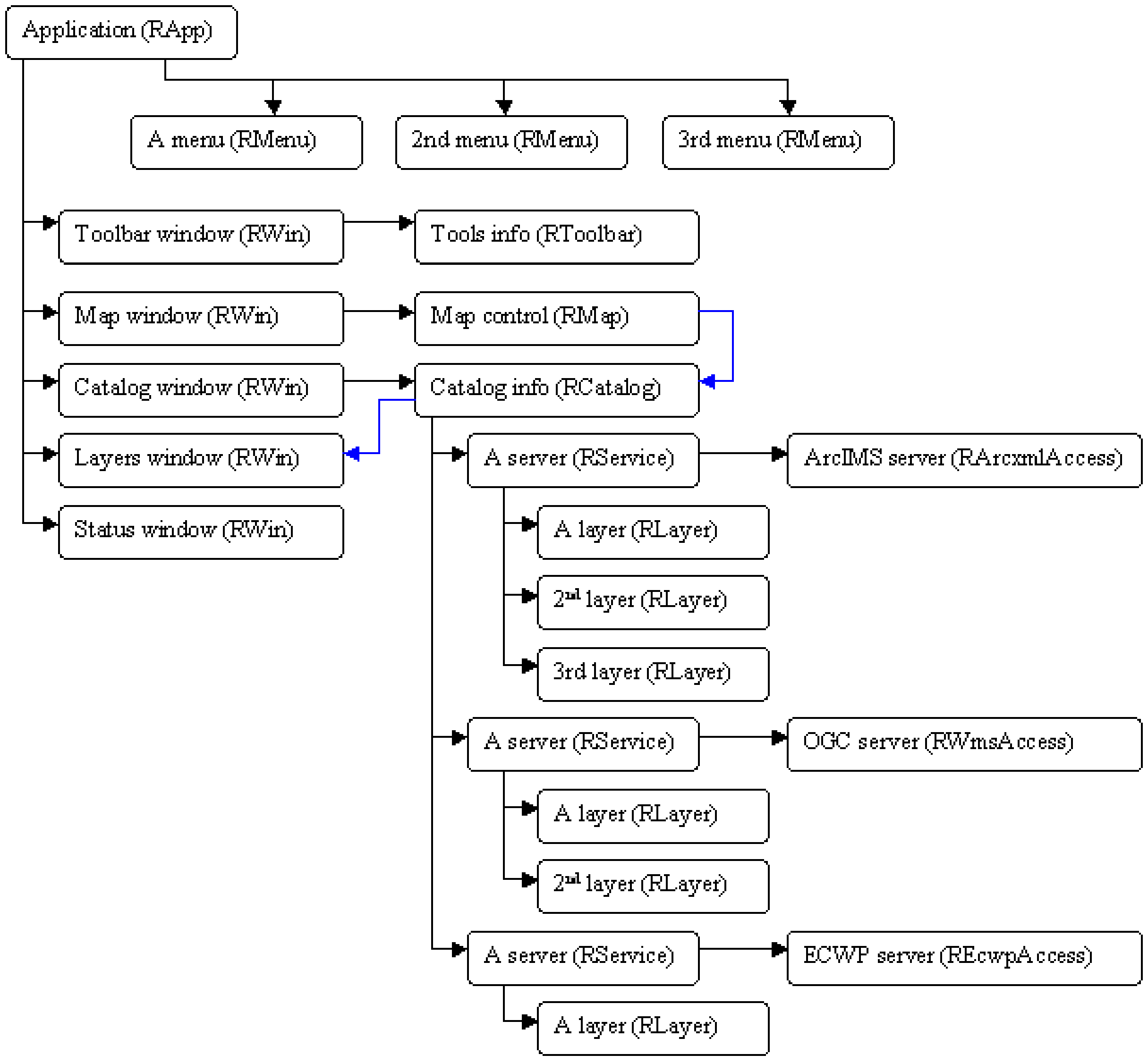
Naming conventions

Objects class names start with the capital letter "R" for the class definition, and by convention the lowercase letter "r" is used for objects themselves.

Object hierarchy

Objects are managed in a tree structure. For example, a typical map page will have the following object layout, with a description for each object and class name.

Black lines indicate object **ownership**. Blue lines indicate object **attachment.**



Object ownership and the .add() methods

All objects must be "owned" by a single parent object (except the root rApp object which has no owner).

RightWebMap objects (such as RCatalog class objects) have one or more **.add<type>()** methods to create and add children objects to a parent.

Add methods may be called by using:

```
object.addObject(new child object);
```

or by calling add methods which create and add a new child. The type of child added depends on the name of the add method, for example:

```
rToolbar.addTool( "parameter",….)
rCatalog.addService( "parameter",….)
```

Examples:

Add a toolbar icon to a toolbar object:

```
rTool.addTool("Zoom","zoom",true,"mapZoom");
```

Add a service to a catalog, then add several layers to the service object:

```
var imageSvc  = catalog.addService("Images","ecwp",0,0,100,true);
   imageSvc.addLayer("W.A. Geology", image1 ,false,false);
   imageSvc.addLayer("Topo","topo"    , image2 ,false,false);
   imageSvc.addLayer("Landsat","landsat", image3,true,false);
```

Adding objects is based on a logical basis – only certain objects can be added to other objects.

As a general rule of thumb, RWin and RMenu class objects can be added to the root rApp object. RCatalog, RToolbar and RMap objects (which are displayed in some way within windows) can be added to RWin objects.

RService objects are added to RCatalog objects, and RLayer objects are added to RService objects.

Attaching objects using the .attach() method

An object can only have one parent object. This means a parent object, such as a catalog, can have several child objects, such as services, but a service can only be attached to one catalog.

It is often desirable to associate, or attach, an object to another related object. An example of this is attaching map objects (which display maps) to a catalog object (which contains the information about services and layers within those services needed to draw a map).

The **.attach()** method is to form these associations.

Only certain associations make sense. Attempts to attach an object to another object that is not related will be ignored.

Common attachments are:

Attach two maps to a catalog:

```
rCatalog.attach(mainMap);
rCatalog.attach(overviewMap);
```

Attach a window to a catalog. The window will be used to provide information about the currently selected service; the catalog will attach specific services to the window for display.

```
rCatalog.attach(rWin);
```

Attach a service to a window that always shows information for that specific service:

```
rWin.attach(rService);
```

Attachments are object specific; for example attaching a map to a catalog enables the two objects to exchange information between themselves. Specifically, a map object uses the attached catalog object to get information about which services and layers within the services to display. A catalog object will call map objects attached to the catalog whenever, for example, a service or layer is turned on or off.

Attach method typically follow the one to many rule. For example:

- Maps are attached to catalogs (as one catalog can relate to many maps)
- Services are attached to windows

Some objects allow multiple attachments; other objects will only use the most recent attachment.

Attachment results are specific to the objects being attached.

The main object classes

The main classes are as follows:

Summary

RApp	The main application class, there is only one rApp object created from this class, and the rApp object always exists.
RWin	Window layout class. These objects are used to define windows (which might instead be panes in a Frame style layout). The RWin objects do not define what the contents are of each window; they simply define the existence of windows.
RMenu	Menu class.
RCatalog	Catalogs contain a collection of services (map servers) that can be shown in a map.
RService	A Service is a single web map server host. A service will typically have more than one layer of data that it can serve.
RLayer	A map layer within a service.
RSearch	Services can have one or more search capabilities associated with that particular service. For example, a service might be able to search a map location based on street address, or on property folio number. An rSearch object is defined for each search for each service. Some services automatically build search capabilities based on information returned from the service during capabilities query.
RMap	Map objects used to display maps within windows. So there will be one rMap object per rWin displaying a map. Put another way, if there are two windows being used to display maps, then there will be two rMap objects, one for each window. Map objects are attached to Catalogs which contain details about the services (web map hosts) used to display maps.

RToolbar	Most web maps will have a toolbar window to allow the user to interact with the map. An rToolbar object will be added to one of the windows to be used as a toolbar.
RStatus	The status line object added to status windows. This is attached to the main map object for the page.
RAccess	Different types of map servers (for example ArcIMS using ArcXML or MapXtreme or MapPoint .NET) have different ways to access the map server. The RAccess object provides a generic interface to map servers. Each rService object defined must also add a rAccess object to access that particular map server. When a new RAccess style object is created, it actually returns a sub-class object specific to the map server being used. These subclasses implement the same methods as the RAccess class in ways specific to each map server being accessed.
REcwpAccess	Interfaces to ECWP (Image Web Server) web servers that provide streaming imagery support. A subclass of the RAccess interface class.
RArcxmlAccess	Interfaces to map servers such as ESRI’s ArcIMS that use the ArcXML style interface.
RWmsAccess	Interfaces to OGC style Web Map Server (WMS) servers that use the WMS style interface.
RMapServerAccess	Interfaces to MapServer servers, which is a free and powerful map server product.
REview3Access	Interfaces to EView version 3 style map servers. EView is a proprietary map server developed by ESRI Australia.
REview4Access	Interfaces to EView version 4 style map servers. EView is a proprietary map server developed by ESRI Australia.
RMapPointAccess	Interfaces to Microsoft MapPoint .NET style map servers.
RMapxtremeAccess	Interfaces to MapInfo MapXtreme style map servers.
RMapguideAccess	Interfaces to Autodesk MapGuide style map servers.

Each object may have different methods. Public methods are detailed as follows:

Rapp methods

.

`latitudeToString(dd)`
`longitudeToString(dd)`

Converts a numeric degrees value into a displayable string in the DD:MM:SS.SSN format.

`stringToDegrees(s)`

Converts a string with degrees (which may be decimal degrees in the form dddd.ddddd or a formatted Degrees:minutes:seconds format) into a number.

Rwin methods

.

RMenu methods

RCatalog methods

RService methods

RLayer methods

RSearch methods

RMap methods

`setExtents(tlLatitude, tlLongitude, brLatitude, brLongitude )`

Set top left and bottom right extents for the map.

`setCenter(latitude, longitude)`

Set the center location for the map, at the current scale

RToolbar methods

RStatus methods

RAccess methods

All the Access objects (REcwpAccess, RWmsAccess and so forth) have the same methods:

Object startup and processing workflow

Once RightWebMap objects (such as windows, menus, maps and so forth) are defined for a page, the page can then be initialised and built.

RightWebMap automatically handles this process for you, however it can be helpful to understand the process.

In particular, several form variables are automatically tracked and extracted for your use. These include:

- The current style
- The optional current XML parsing string used to create the page
- The optional maps to use for this map

Use the *rApp.loadPage(style,xml,maps)* method to reload the map page with a different style, different xml control and/or different maps to display.

The following values are available for your use:

```
rApps.style            // current style in use
rApps.xmlUrl           // current XML path
rApps.maps         // maps to display
```

For example, your HTML code loading services could load different servers based on which maps the user wishes to see, as follows:

```
if(rApps.maps == "usa" ) {
   // add some services specific to the USA
}
else if( rApps.maps == "australia" ) {
   // add some services specific to Australia
}
```

Swapping styles, xml control file, or maps

You can allow the user to swap to new styles or XML control files or maps. The *rApp.loadPage(style,xml,maps)* method can be called with any of the parameters missing or null, in which case the current values are used. This is handy creating menu items that are only changing one thing.

The init() method: initialising objects before page is displayed

Once objects are created, they are then initialised. For this reason, most objects have a.init() method. Objects are initialised *before* the page body is started. Some objects, in particular windows and menus, will emit HTML code during the init call to define the object outline on the page.

Code generated during the init() method can not be changed at a later time; it is not dynamic, so it should only be used for static output of code; typically menus and backgrounds.

Objects that need to be able to modify the contents of a window (or menu) object at a later time should put the HTML code in the build() method instead of in the init() method.

When objects are created, they should themselves to the master list of RightWebMap objects, using:

rXadd(this);// add object to master list

The object can then be identified in DHTML by using *this.myself* which will return a pointer to this object.

The build() method: building contents after the page is displayed

Many windows have dynamic content. That is to say that the contents of the window can change *after* the page has been displayed.

An example of such dynamic content is the layer list for map servers that are queried to report what layers are supported by that service, and the service information is then built after the server has responded. Another example is dynamically created search windows.

Once the init() phase is finished, and after the body is ready for display, objects are then built using the *.build()* method. This method typically constructs the actual contents for windows. The *build()* method for an object should only be called by the window that was the parent to the object or that has been associated with the object.

Some objects may not even display contents initially, or the contents may change and then may need to be later rebuilt.

Such objects have a *update()* method, which can be called to update the object if it is currently associated with a window. You can't directly call the *build()* method for non-window objects; use the *update()* method instead, which will first check that the object is indeed associated with a window in some way.

Internal map window design

RightWebMap also makes use of a map plug-in which manages the internal issues of gather map data from different map servers and integrating all the map layers into a single map view. The map plug-in used depends on the browser and operating system, and may be a native plug-in (Windows or Macintosh) or a Java plug-in.

Creating different styles and defining window sizes

RightWebMap enables different styles to be easily added or changed. Styles are independent of the underlaying map application; different users of the same web map can use different styles. A user can even swap between styles. Two styles are provided as standard, the **Win** style and the **Frame** style.

Window styles: Frame versus Win style

Web pages built with RightWebMap can have different styles.

You can customize styles, and two styles are supplied as standard: the "Frame" style and the "Win" style. These reflect the two basic layout styles that a RightWebMap authored web page can have. Your users can swap between styles if you allow them to.

Some window browsers (notably ones that use Netscape plug-ins or Java plug-ins) don't allow map windows generated by the plug-in to be layered "behind" other sub-windows on the web page.

For these types of browsers, your users will need to use a "Frame" style.

When you create new styles, any style starting with the keyword "Frame" is automatically assumed to be of the Frame style.

Windows within pages can include windows like status bars, tool bars, map windows, catalog panes, and so forth.

When the web page is resized, the RightWebMap authored page may resize some of these windows automatically.

Some applications may only connect to a few services. In these cases, you may choose to turn off the Catalog pane. Note that it must exist (as it contains the catalog), it is just that it does not have to appear on the web page.

Features specific to Frame styles

Specifically for Frame style layouts, if all "pane" control windows are turned off, then the map is automatically resized to take advantage of the additional available space.

Also, frame "pane" windows are resized automatically as ones are turned on or off.

Finally (again only for frame styles), if the user makes a service active, and that service is attached to a window pane but the window pane is currently closed, it is opened.

Win style

This style will look familiar to Windows users. Map windows within the browser page can be moved, resized and iconized or closed.

Frame style

This style implements a typical older style web map "frame" interface, with each window in a non-overlapping layout.

Using styles

When writing new RightWebMap objects, you will sometimes need to know how to access the current style information.

Two variables are useful:

```
rApp.iStyle             // the directory containing this style's images
rApp.cStyle             // the name of this style in the CSS file
```

Style implementation issues

New styles can be added by adding the new style(s) to the RightWebMap.css file, and by adding appropriate icons to the appropriate image style directory. For example, the standard implementation has two style image directories: WinStyle and FrameStyle.

Some browsers and some operating systems have restrictions that prevent the use of overlapping windows for the user interface.

Specifically, browsers that must use the Java map plugin or the Netscale style plugin can not have windows overlap map windows (this is because Java and Netscale plugins don't allow other objects to appear over the plugin windows).

For these browsers, Frame style interfaces should be used.

RightWebMap automatically detects if the browser has these restrictions, and uses a Frame based style for those browsers.

Changing styles

Style can be changed by reloading the web map page with a new style.

For example, the following adds a menu with two menu items to change the style for a page:

```
varstyleMenu = rApp.addMenu("Layout Style");
      styleMenu.addItem("Windows" ,
              'javascript:rApp.loadMap("Win","' + rApp.xmlUrl + '")' );
styleMenu.addItem("Frames"
,'javascript:rApp.loadMap("Frame","' + rApp.xmlUrl + '")'  );
```

This example uses the rApp.loadMap(style,mapXml) call to reload the page with a new style and with the current map definition XML.

Creating new styles

New styles can be added by adding the new style(s) to RightWebMap.css and adding a new image style directory.

If your new style will have a frame style layout, it must start with the word "Frame" so that RightWebMap knows that the style is a frame style layout.

All other styles are assumed to be resizable window styles.

Fixed versus scrollable portions of windows

Windows or frame panes can be resized. Contents that are larger than the window area automatically have scroll bars added when required so the user can scroll through content (such as layer lists) that may be larger than the window area.

However, some objects may require both a fixed area and a resizable area. When a window object is created, an optional fixed height area can be specified. This area is optional; if the parameter is not specified then the window will not have a fixed area and the fixed area must not be used.

The fixed portion of a window is always immediately just above the scrollable area. This means that the fixed area can be used for things like tabs, which can then direct the scrollable area to change based on the currently selected tab.

Note: Some windows, specifically status and toolbar format windows, will ignore the fixed height area even if specified as these types of windows have a specific layout.

Defining window size hints for objects

Objects can specify hints for their preferred width and height. These are only hints – the window style can (and generally will) override hints, depending on style (framed versus window styles), users resizing windows, and/or browser window size.

The way to set window size hints is for the object to see if it has a rWin handle, and if it does, then the object can modify the window height or width values.

Note: The only object that is likely to need to do this is the toolbar object.

The ToolBar object also modifies the rApp.paneTop variable to take into account the tool bar size.

Setting window titles

It is sometimes desirable to be able to change the title of a window. Window objects have a *setTitle(newTitle)* method to change the window title.

Some window types may ignore the title and not set it. These windows include status windows and toolbar windows.

Laying out tool bar items

Tool bars appear differently under *Win* based styles compared to *Frame* based styles.

Under *Win* based styles, the tool bar is a non-resizable window, two icons wide and as long as required to fit all tool icons.

Under *Frame* based styles, the tool bar is at the top of the page.

Sometimes you may want to insert spaces into tool bars, to logically group icons, or to have multiple rows of icons on frames.

Spaces can be added with the rToolbar.addSpace() method, which can be passed "Row" or "Column" to indicate a new row or a column space.

Note: To conditionally insert a space depending on style, you can check the *rApp.framed* value, for example:

```
if( rApp.framed )
   rToolbar.addSpace("Row");
```

Making service layers colors transparent

It is often desirable to make certain colors within a service transparent.

For example, a service may use white as a background color. This could be made transparent so that underlaying services show through.

Each color in a service can be made transparent, or semi transparent.

The toolbar Colour pointer item can automatically make colors transparent.

In addition, services can have default colors made transparent when the service is initialised. These are marked as "system" colors to ensure they always remain transparent.

Methods used to alter color transparency

Two *rService* methods are useful when knocking out colors:

```
rService.hideColor(color, percent, scale, userColor);
```

This method provides complete control over a specific color. The color transparency can be specified, as can a map scale to remove the color, and if the user can modify setting.

The following call is primary used for pre-loading several colors. It always specifies full transparency, no scale, and as a system call:

```
rService.hideColors(colorArray);
```

For example:

```
rService.hideColors(["#ffffff","#ff0000","#ffff00"]);
```

Will hide white, red and yellow.

Working with servers that return JPEG images

Ideally, a server should return a GIF or PNG image. However, some servers can only return JPEG images.

This causes a problem, because the compressed JPEG changes a single map color into a spread of similar colors during the compression process.

A mask can be specified to "round" color values using masking. The value used to set this is:

```
rService.colorMask = 0xe0e0c0;        // colors will be &'d with this mask
```

Adding panning and zooming

Setting map location

A service might perform a search, and want to update the extents or center point for a map.

Methods exist for the rMap object to set the center point using the current scale, and to set the extents for the map.

Units are always in Latitude Longitudes; the rMap object converts these to the actual map units (for example easting northings) if required.

Adding real time panning

Some services, such as services that are providing real time feeds from GPS unit locations, will want to update the map location in real time to follow the unit as it travels across the map.

If a service wants to be able to update the map location in real time, it should check the *rMap.realtimeRoam* value is set to *true* before doing a *rMap.setCenter* call.

This is so users can turn off real time panning when they need to manually control map location. Typically, the users can turn real time panning on or off using a toolbar option.

Adding query capabilities

RightWebMap has full query capabilities. Users can:

•Query one or more map servers – at the same time

•Specify which layers within map servers are to be queried

•Queries can be returned from map servers even if they are not visible on the map

Queries can be formed in the following ways. The italic name is the name for that specific query when adding tools to the toolbar.

•*mapQueryPoint (*#1)
Point location queries (return information for a specific map location)

•*mapQueryRectangle* (#2)
Rectangle location queries (return information for rectangular area in a map)

•*mapQueryPolygon (*#3)
Polygon location queries (as above, except for an arbitrary polygon)

•*mapQueryCircle* (#4)
Circle location queries (as above, except for a circular area)

Query capabilities are defined for layers for a map service if the interface protocol for that map service sets the *canQuery* variable to true.

The service can decide if it is capable of doing queries based on initial configuration information from the map server. For example, a protocol might be capable of performing queries, but a specific map service might not offer query capabilities.

The *Query* tab is visible for map services that can perform queries; if the tab is not present for a server (because the *access.canQuery* variable is *false*, that service can not perform queries).

RightWebMap manages the user interface for queries using a vector map layer called *editLayerName* within a map object.

The vector edit map layer is turned on and off by RightWebMap as appropriate during queries.

After the user has specified the query area, each service (that supports queries) is called with the query information. The services then perform the queries, and return the results, shown in the query tab.

Access interface protocols should inspect the *rMap.doQuery* value to determine the type of query to be performed. If the access interface can't handle a type of query, it should at least attempt a lesser form of query. For example, if an access protocol can't do a circle query, it should use the center point of the circle and perform a point query.

Adding search capabilities

Searching is a critical portion of web map pages. For example, your clients may need to be able to zoom the map to street address.

The RSearch object is used define search capabilities in a generic way.

A given rService (from the RService class) may have one or more search capabilities.

In order to be able to be able to search, a service must:

- Have the service.access.canSearch set to **true**
- Have at least one search object

Adding more than one search capability

The following example shows two manually created search objects added to a service (in this case the cadSvc service).

Note: Many services have the ability to construct search operations after querying service capabilities; the following example is instead a manually constructed example.

```
search = cadSvc.addSearch("Address");
   search.addField("Name");
   search.addField("Street","street",30);
   search.addField("Suburb","suburb",20,"Nedlands");
search = cadSvc.addSearch("Folio");
   search.addField("Number");
```

Note that each search capability is added (the addSearch("Address") and addSearch("Folio") objects in this example), and then fields within each search object are added.

The RightWebMap user interface automatically present search capabilities to the user in a reasonable way based on the search information.

If there is only one search capability for a service, it will be shown as a single search capability for that service under the Search tab. If there is more than one search capability, then at the start of the search there is a drop-down multi-choice list allowing the user to select from different searches to use.

Different search fields

A search object can be created manually or automatically from a server reporting its search capabilities.

A search can have many fields, such as number fields, text fields, tick boxes and multi-choice fields.

All *add* methods start with the name for the field to be displayed for the user as the 1st parameter, and the key for the field (normally passed to the map server as an ID) as the 2nd parameter.

The key field is optional, in which case the key will be set to the name.

Note that a method supports additional arguments and that they are specified, then the key field has to specified.

Text fields

This allows an arbitrary string field to be entered for the search. Syntax is:

```
rSearch.addString(nameString [, keyString, lengthNumber, defaultString] );
```

Specify the field name (*nameString*) and optionally a key name, length and a default, for example:

```
rSearch.addString("Name", "namefield", 20, "Renalds" );
rSearch.addString("Address");
```

Multi-choice fields

Allows an multi-choice field be entered for the search. Syntax is:

```
rSearch.addChoice(nameString, [keyString, choiceStringArray ,defaultString] );
```

As with other fields, the *nameString* is the name for the field. The default choices array is ["Yes","No"].An optional default can also be specified.

The array of choices can be created using normal JavaScript array initialisation.

```
var choices = [ "Monday", "Tuesday", "Wednesday", "Thursday" ];
rSearch.addChoice("Day", "selectday", choices);
```

The above example creates an array of choice strings and passes that array as the choice array. The following examples all generate the same results, as they build on default arguments:

```
rSearch.addChoice("Today");
rSearch.addChoice("Today", "Today");
rSearch.addChoice("Today", "Today", ["Yes","No"], );
rSearch.addChoice("Today", "Today", ["Yes","No"], "Yes" );
```

Numeric fields

Accepts entry of numbers only:

```
rSearch.addNumber(nameString [, keyString, defaultNumber] );
```

Specify the field name (*nameString*) and optionally a default, for example:

```
rSearch.addNumber("Street Number" );
rSearch.addNumber("Address");
```

Tickbox true/false fields

These allow uses to specify a tick (yes/no or true/false) option field:

```
rSearch.addTick(nameString [, keyString, defaultBool] );
```

If not specified, the default is *true*.

Specify the field name (*nameString*) and optionally a default, for example:

```
   rSearch.addTick("Detailed search" , "detailkey", false );
```

Execution of searches

When the user has entered a search, it is passed to the RAccess interface object specific to that service with the map to search and the search object to use, for example:

```
RAccess_SearchMap(rMap,rSearch)
```

The reason that the rSearch object is passed (instead of just using the service.currentSearch index) is so that RightWebMap applications can construct search objects on the fly and pass them to the service access interface for advanced on-the-fly search operations.

The access method must execute the search, then zoom the map to the appropriate area.

Variables present within search fields:

The following fields are present within search fields:

All field types have the following fields:

style	String value, one of: "string", "number", "choice" or "tickbox"
name	Displayed name for the field
key	Key for the field. By default same as *name*
value	The selected value. May be a string (for "string" or "choice", number (for "number", or a boolean (for "tickbox") value.
uid	A unique ID that will be different for every search field across the application (often used for creating field unique HTML code)

Depending on the style for this field, the following may also be valid:

length	"string" fields only: the length of the field
choices	array of strings for multi-choice

Smart management of layers

A service can have multiple layers. It is possible to control how and when layers are displayed or hidden, based on:

•If a layer group is on or off (some services can group layers)

•If the scale factor is too large or too small to display a layer

•If the extents of the current map view are totally outside the extents for a layer

•If the layer is an Exclusive Or (Xor) layer. For, if the 2001 layer is displayed then the 2002 layer may be turned off using this.

•If the layer is in a different map projection. *RightWebMap* also handles changing map projections on the fly to manage different images in different map projections. This is implemented using the Xor capability above.

Any or all of the above capabilities can be used to manage layers. When layers are outside the "display" rule set, then they can be completely hidden from the layer list so the user only sees layers appropriate to their current map view.

In the simplest case, if none of the above are set for layers, then layers are managed as a simple list. The user can still turn layers on or off in this set.

Note that the concept of *hiding* a layer is different from if the layer is turned on or off.

A layer can be turned on, but hidden, in which case it will not be used or displayed in the layers list.

A layer can be turned off, but not hidden, in which case it will be displayed in the layers list for the user, but won't actually be displayed until the user turns it on.

Grouping layers into groups

Some – but not all - services can group layers into groups.

If a service can support layer groups, then the access interface must support this capability when working with layer lists for a service.

It is possible to implement a service without grouping.

It is also possible to support grouping (to make layer lists easier to manage for the user), but hide this from the actual service itself by collapsing the groups into a straight list when interacting with the service from within the access interface protocol object.

One common – and powerful – use for grouping is to group an entire set of images together as a single group.

For example, a state might have 100 cities and towns, each with their own high resolution airphotos. The airphotos might be available for 3 years, for example for 2001, 2002 and 2003.

A simple way to manage this is to group the photos in to years. If extents and/or scale are also set, then the user will only see layers that are appropriate to their current map view.

Layer groups can be cascaded, with groups of layers in turn containing other groups of layers.

Hiding layers if scale is too small or too large

Layers will automatically hidden if the current map scale is outside larger or smaller than the values specified in the setScale method:

```
rLayer.setScale(minScale, maxScale);
```

If minScale or maxScale is zero, then that test is not performed (so you can specify a minimum scale, or a maximum scale, or neither, or both).

The default is zero for both scales on new layers, so this test will not be performed by default unless scales limits are explicitly specified.

Scale is expressed in degrees across the current map view.

If the layer is outside the current scale, then it is *hidden*.

Note that even if current scale is within range, the layer may still be hidden for another reason (such as it being completely outside the current map view extents).

Hiding layers if completely outside the current map view

Layers will automatically be hidden if the layer's extents are *completely* outside the current map view. Use the following layer method to set extents:

```
rLayer.setExtents(tlLatitude, tlLongitude, brLatitude, brLongitude);
```

The test to hide a layer based on extents will only occur if any value is non-zero.

The test will not be performed if all values are zero (the default for new layers).

Hiding layers using "Xor" Exclusive-Or capability

It is often desirable to hide a set of layers if another set is activated. One reason for doing this is ensuring only layers in the same map projection are visible.

The following call sets the exclusive name for a layer. RightWebMap ensures that only other layers (in the same service) with the same Xor name are turned on at the same time.

rLayer.setXor(name);

It is possible to have a conflict where more than one set of Xor names are valid for the current scale and layer extents.

In this instance, the valid Xor group is taken from the first top-most layer that is not hidden by scale or layer extents, ***and covers at least 3 sides of the current map view.***

This way, smaller map views that are too small to cover the map area are not used until the map is zoomed into that area, even if the map layer is a higher priority map layer than a layer covering a larger area.

For Xor group searching, the list is searched in the *creation* order, not the current user selected layer ordering (to ensure that the predefined Xor grouping selected for the service remains constant).

In other words, the most desirable layers (e.g. highest priority in the Xor selection) should be added to the layer list for a service first.

Hiding layers in incompatible map projections

A typical map may have a large number of images that can potentially be displayed. For example, the following images might be available:

1.Large scale images, covering the entire state, in a Geodetic map projection, such as:

a.DEM

b.Satellite imagery

2.Year 2001 High resolution images covering each city or town

a.City 1

b.City 2

c.City 3

d.City 4

3.Year 2002 High resolution images covering each city or town

a.City 1

b.City 2

c.City 3

d.City 4

e.City 5

and so on.

Typically, the imagery for each city might be in a UTM zone rather than in Geodetic. This means that images from one zone can't be displayed beside imagery in the next zone, as they are not Geodetic. (Note that this restriction is being removed in a future release).

Also, when the user zooms out a larger view covering a large portion of the state, it is desirable to swap views to using the satellite imagery covering the entire state (but at a lower resolution than the high resolution airphotos).

It is very easy to manage the above, by using two rLayer methods:

```
   rLayer.setExtents(tlLatitude, tlLongitude, brLatitude, brLongitude);
   rLayer.setXor("UTM54");            // only compatible images can also be
displayed
```

Using the above two methods on each layer (and possibly the *setScale()* method as well), *RightWebMap* will automatically manage your images, ensuring that only images in a compatible map projection will be displayed at the same time.

Note that the *Xor* method, here used to name the map projection used for each image, crosses layer groups. In other words, it is applied after considering other rules.

Integration issues with specific web map servers

Working with servers that return JPEG images

Some servers return JPEG images instead of PNG or GIF images (which are preferred as they are smaller for vector style maps).

It is difficult to remove colors from JPEG image map layers, as a single original map color may be converted to many colors due to JPEG compression.

Where a server can only return JPEG and not some other preferred image style, a *colorMask* can be specified for that service, which is used to as a mask during map color transparency calculations. The mask is specified as a 24 bit hexadecimal number. It should be set for the service before any color operations, such as *hideColor*, are performed. For example:

```
eviewSvc.colorMask = 0xf0f0f0;
```

Working with ArcIMS servers

Working with MapServer servers

Working with OpenGIS OGC WMS servers

WMS Version 1.1.0 or higher are supported

RightWebMap supports WMS servers that are version 1.1.0 or higher.

This is because there were substantial changes from the 1.0.0 WMS servers (which were quite limited) and the later versions.

Should it be essential to support 1.0.0 WMS servers, it would be possible to add an interface protocol for earlier version servers.

Passing host and service details to a WMS server

The concept of "host" and "service" for WMS servers can vary from server to server.

In general, ArcXML based servers that are serving WMS often use a parameter:

```
?ServiceName=ThisServiceName
```

although ServiceName= is not a formal WMS parameter.

In the same way, MapServer based servers that are serving WMS often use a parameter:

```
?map=/path/to/mapfile.map
```

Finally, another WMS server may not need any map parameter at all. For GET based WMS servers (most at present), this causes problems with parameter passing.

For the above reasons:

•Specify the service name parameter (such as ServiceName= or map=) as part of the host URL

•Specify just the service name as the service parameter

Configuring WMS servers to handle many layers

Most WMS servers currently only use the GET method of access. This restricts the size of the URL that can be passed to the server.

It is *vital* to keep layer names as short as possible in order to avoid exceeding probably URL length limits.

Note that WMS servers can return the layer "name" which is meant to be machine readable and a short identifier, and the layer "title" which is meant to be human readable and can be longer.

Many WMS servers only return the layer "title" , which must then be used as the layer name, which can cause URL lengths to be quickly exceeded.

RightWebMap's WMS interface access methods limit the number of layers that can be turned on or queried in order to avoid this problem.

Ensure WMS servers return "correct" XML

The XML standard says that XML parsing is a pass/fail process – if it is not all perfect, then XML parsers will simply return an error and fail.

This was designed into XML to prevent problems where badly formed pages had to be somehow interpreted (which was a major problem with HTML and one reason why different browsers react differently to badly formed HTML).

Many web map servers that are configured to serve WMS are generating invalid XML. Technically, this should cause the service to fail.

RightWebMap tries to parse XML files to recover from most of the obvious errors caused (particularly from ArcIMS based servers hosting as WMS servers).

However, the real solution is to ensure that your WMS servers generate correct XML.

Adding new interface classes for different map servers

RightWebMap comes standard with interface classes for many commonly used web map servers, including ArcIMS(ArcXML), OGC (WMS), Mapserver, MapXtreme, MapGuide, MapPoint .NET and streaming ECWP imagery using the Image Web Server.

Map layers from any or all of these types of servers can be stacked in a single map.

Should support for other map servers be required, new interface classes can be added to support the new server.

To add support for a new service type, a new class and some associated methods must be created.

Note: you only need to add a new access class if you are adding access to a new service protocol that is not already supported by RightWebMap (e.g. once access has been added, it can be used in any map from then on to access any servers using that protocol).

Interface classes are named **R*Interface*Access()** where ***Interface*** is the name of the new interface method.

Two lines must also be added to the **RAccess()** class use the new class when required. For example, the following 2 lines are used to access OGC/WMS style map servers:

```
   ...
else if( interfaceName == "wms" )
   access = new RWmsAccess(service);
...
```

Hosts and services

Many map servers have the concept of a host, which has different maps, or services, that can be accessed (and typically layers within those services).

Any new service interface added has access to the host and service name defined to access a specific service. These and other information are held in the RService object which is associated with a specific interface access method.

Accessing a map server

So long as a map server can return a image in a format such as PNG, GIF or JPEG for a requested area, it can be integrated into a RightWebMap map page.

Some servers also allow the list of layers to be queried and returned, or the available search methods to be queried and returned.

These and other capabilities can be built into a server access class.

Methods required for new interface access classes

The new **R*Interface*Access()** has a number of mandatory methods that must be added, as shown with the following hypothetical Acme map server interface protocol:

```
//  ---------------------------------------------------------------------------
// Acme interfaceName access
//  ---------------------------------------------------------------------------

function RAcmeAccess(service) {
   this.name = "Acme Map Server";
   this.version = "1.0";
   this.orderLayers = RAccess_OrderLayers;
}

// most servers can use the default Raccess_UpdateMap method
RWmsAccess.prototype.updateMap = RAccess_UpdateMap;

// method to request a map from the server
function RWmsAccess_LoadMap(rMap)
{
   // add logic here to load a map
}
RWmsAccess.prototype.loadMap = RWmsAccess_LoadMap;
// method to request a list of layers from a server
function RWmsAccess_LoadLayers(rMap)
{
   // add logic here
}
RWmsAccess.prototype.loadLayers = RWmsAccess_LoadLayers;

// method to handle the map (or layer lists or search) response from the server
function RWmsAccess_ResponseMap(rMap, layerName, url, body, action, tlx, tly, brx,
bry, response)
{
   // add logic here
}
RWmsAccess.prototype.responseMap = RWmsAccess_ResponseMap;
```

Deciding size of map to return from a service

When an access method is called, it is handed the rMap object being used to receive the service response image as a map layer.

There are two variables:

- rMap.width

- rMap.height

which can be used to specify the size of the returned map image.

Many servers have limits to the size of the map that can be returned (typically the image being generated must be less than 1024 x 1024 in size).

There are two rescaled variables which can be used instead, which will always be less than 1024 x 1024, but are correctly scaled to window size. These are:

```
rMap.width1K
rMap.height1K
```

The RightWebMap map control object, which manages map layers, will automatically rescale the image up to the appropriate window size, so it is acceptable to request and receive a map layer that is smaller than 1K x 1K even when the map window is larger than this.

Note: most of the standard service interface objects limit the server requests to the above limit to ensure robust operation in all situations.

Enhanced functionality for interface access classes

Services will query interface access classes for additional capability. If present, additional functions will be shown on service windows and made available if the following are set:

```
RAccess.hasLegend = true;          // protocol can return a legend
RAccess.hasSearch = true;          // protocol can perform searches
RAccess.hasQuery       = true;         // protocol can return layer query info
```

Implementing query capabilities

If an access method has the *hasQuery* value set to true, then it will be called when a map query is performed by the user.

The RAccess_QueryLayers(rMap) method is called to perform a query.

The rMap object has several values of interest:

The *rMap.doQuery* value contains the type of query as an integer, being:

1- point query

2– rectangle query

3– polygon query

4– circle query

If the interface access can not handle all types of queries, it should still perform some query. For example, if the interface can not perform a circle query, the center point of the circle should be used and a point query performed instead.

The *rMap.queryPoints[]* array is an array of points for the query. Each point object contains the latitude, longitude, and also worldX and worldY values for that point. Some map servers require query locations to be passed as pixel locations for the current map view. The point array also contains screenX and screenY for these servers, however these values are only valid during the query itself (they will be invalid once the user roams or zooms, although the other values will remain valid).

Point queries have one point, being the point query location.

Rectangle queries contain two points, being two corners of the rectangle.

Polygon queries contain a series of points, one per point in the polygon. The first/last point is only specified once, as the first point in the array.

Circle queries contain two points, the first point is the center of the circle and the second point is an edge location for the circle.

Handling XML: the RXml object

Many servers communicate using Xml. This is particularly true when requesting layer information and/or search capabilities from servers.

Typical usage

Different browsers handle Xml in different ways. To hide this, there is a RXml object that can be used. Typically it is accessed within access interface object.

A new rXml object is created and used as follows:

```
this.rXml = new RXml();                      // create the object
this.rXml.loadXmlText(response);             // load it from text
// try and get an entry we need
var node = this.rXml.rootXml.getElementsByTagName["WMT_MS_Capabilities"][0];
if( node ) {
   // process it
}
```

RXml loadXmlText and loadXmlUrl methods

Once created, the rXml object contains Xml DOM object, as the "rootXml" object within the rXml variable. This is a subsidiary object, because it may be an ActiveX object under IE (using Microsoft’s XmlDom ActiveX object).

The rXml object has several methods to load and manage XML from string responses from servers and from Urls, as follows:

```
   rXml.loadXmlText(txt,[strip]);         // loads the XML object from a text
string
```

```
rXml.loadXmlUrl(url,[strip]);             // loads the XML object from a URL
```

These hide the different ways that these functions are implemented in different browsers.

Browser and XML server specific issues

The RXml object hides browser specific issues from your code.

Some map servers don't generate correctly formed XML (for example they may have invalid trailing zeros after the XML data).

The RXml object turns off strict XML checking so that more robust XML reading can be performed, however some of the stricter XML parsers (notably Microsoft's) will generate errors when parsing some XML from some servers. For robustness, your code may want to trim invalid data with the following JavaScript before using the loadXmlText() call:

```
var index = response.lastIndexOf(">");
if( index < response.length-1 )
   response = response.slice(0,index+1);
```

Some browsers (notably Netscape) insert objects into the Xml tree for lines breaks between Xml lines, for example:

```
<LAYER>
   <NAME>
```

would have a comment object to indicate the newline and spaces after the <LAYER> definition and before the <NAME> definition.

This is seldom needed, and varies between browsers. By default, these are stripped out of Xml in the loadXml() and loadUrl() methods.

If you *do* need to leave these in for some reason, add the Boolean strip variable set to **false** as the 2nd parameter to the methods, and stripping will not be performed.

Note: Be sure to test your application under different browsers if you choose to set stripping to **false.**

You will typically use the first method, for several reasons:

1.Some browsers pop up load warnings when loading XML from different hosts. This makes the user experience confusing when displaying maps with layers from multiple servers.

2.If you use the rMap object to send and receive map requests to servers, the page will automatically indicate when servers are busy or offline.

XML Performance

The process of creating an RXml object (which is really the native XML object for the browser) is a reasonably "heavy-weight" process. For example, under IE browsers an ActiveX object must be created.

Furthermore, the processing of an XML response from a server into an XML object can be quite slow (in the order of 1 second for some XML DOM implementations).

It is recommend that objects are created sparingly, and only used when needed.

For example, it is probably better to just parse a return string to extract the image map name, rather than converting the response XML into a XML object, then parsing it – all just to read an image name,

The RXml objects are mainly intended for infrequently used or complex XML objects, such as querying servers for layers or search capabilities, or for reading back location attribute information.

Accessing XML objects

Access XML information using the usual JavaScript DOM commands.

Note that the IE version of XML uses Microsoft's XML ActiveX object, which is not a JavaScript object so won't behave quite as expected in conditional test situations. For example, you can't do if(xmlNode.getElementsByTagName) to test if the object has some function (getElementsByTagName in this example).

To get a node:

var node = xml.getElementsByTagName("WMT_MS_Capabilities")[0];

Note! This is a method, not an array, so use parentheses not square brackets.

Commonly used node attributes:

node.nodeType// type of node

node.nodeValue// value

node.nodeName// name

node.attributes// array of attributes

Working with JavaScript object classes and methods

If you experience with JavaScript has mainly been in creating web pages, the extensive object capabilities in JavaScript may be new to you.

Objects, dynamic HTML and other capabilities used by RightWebMap are covered in two good O'Reilly books:

•JavaScript: The Definitive Guide, 3rd Edition, by David Flanagan, O'Reilly Books

•Dynamic HTML: The Definitive Guide, 2nd Edition, by Danny Goodman, O'Reilly Books.

The following quick overview of JavaScript objects may also help:

Defining JavaScript object Classes

Using objects - particularly in a powerful and flexible language like JavaScript – is easy.

Essentially, instead of handing functions the names of objects to work on, object oriented programming entails defining object classes, which define "methods" on how to operate on objects of that particular object class.

A new JavaScript object is created using the *new* syntax. The following code defines the constructor for a class, called MyTool, and creates an new instance of that class, called myTool.

```
function MyTool( name, width, height ) {
   this.name = name;
   this.width = width;
   this.height = height;
}

var myTool = new MyTool("Pointer",10,20);
```

By convention, class definitions start with a capital letter and objects created from classes start with a lower case letter.

Note the use of "this." to access variables within an object. This example is called with three parameters, and records those parameters in the object.

Defining and using class methods

Methods (similar to functions) can be defined for objects. This is done by creating a function in someway, and using the "prototype" syntax to indicate that an object class has a new method using that function.

For example, suppose we wanted to have a "size" method for the "MyTool" class, which returned the width * height for the object. This would be defined and used as follows:

```
function MyTool_Size( ) {
   return this.width * this.height;
}
MyTool.prototype.size = MyTool_Size;
```

By convention, the method function name is ClassName_MethodName with both the ClassName (MyTool in this example) and the MethodName (Size in this example) having the first letter as a capital letter.

The method definition (MyTool.prototype.size) by convention uses lower case for method names.

Any function can be used for a prototype method, thus the above could also be written as:

```
MyTool.prototype.size = new Function("return this.size * this.height;");
```

However, this is generally slower to execute that defining the function body in the actual JavaScript rather than using new Function().

In either instance, the method can be accessed as follows:

```
var MyTool = new MyTool("Zoom",20,40);// define an object
var response = MyTool.size();// get size
```

Detecting an object's class

It is sometimes convenient to know what class an object is. An object's constructor will return the name of the constructor used to create that object.

For example, many of the RightWebMap object attach() methods need to know what object is being attached. In the following example, the RMap class of object has an attach() method, which checks to see if the item being attached is a RStatus object:

```
function RMap_Attach(item) {
   if( item.constructor == RStatus ) {
      this.status = item;
      item.map = this;
   }
}
RMap.prototype.attach = RMap_Attach;
```

Constructing different object classes in a unified way

If the constructor for an object returns a value different to the this value, that new return value becomes the object.

This can be used to have a single constructor to hide different styles of object classes from higher level objects.

The RAccess object uses this to hide which particular interface method is being used to connect to a service.